Jannasch, Henrik

Daten-Management für Windparks mit Kommunikation gemäß IEC 61400-25

eingereicht als

DIPLOMARBEIT

Hochschule Mittweida

University of applied Sciences

Fachbereich Informationstechnik & Elektrotechnik

Bad Pyrmont, 2009

Erstprüfer: Prof. Dr.-Ing. Thomas Beierlein

Zweitprüfer: Dipl.-Phys. Ing. Claus Vothknecht

Vorgelegte Arbeit wurde verteidigt am:

Bibliographische Beschreibung:

Jannasch, Henrik:

Daten-Management für Windparks mit Kommunikation gemäß IEC 61400-25 - 2009. - 128 S. Bad Pyrmont, Hochschule Mittweida, Fachbereich Informationstechnik & Elektrotechnik, Diplomarbeit, 2009

Referat:

Ziel der Diplomarbeit besteht in der Implementation einer Schnittstelle nach IEC 61400-25 mit einer Remote Field Controller der Firma Phoenix Contact GmbH, der zur Steuerung von Windenergieanlagen verwendet werden kann. In Windparks organisierte Windenergieanlagen verschiedenster Hersteller erscheinen beim EVU, neben anderen verteilten Energieresourcen, als sog. virtuelle Kraftwerke. Ihr zunehmender Anteil am Energiemix macht es notwendig, Verhalten und Einflussmöglichkeiten denen konventioneller Kraftwerke gleichzusetzen. Die Erfüllung dieser Forderung macht die dafür erforderliche Anpassung der bisher ausschließlich verfügbaren proprietäre Schnittstellen verschiedener Anlagenhersteller besonders aufwendig. Die IEC 61400-25 unternimmt den Versuch, eine einheitliche Schnittstelle auf Basis eines objektorientierten Ansatzes vorzunehmen. Nach einem Überblick über diese Norm wird ein Implementationsweg unter Verwendung der hauseigenen Automatisierungslösung PCWORX aufgezeigt. Es wird ein Lösungsweg dargestellt und beschrieben, der alle Aspekte durch Transformation objektorientierter Gesichtspunkte nach IEC 61131 umfasst.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

ACSI	Abstract Communication Service
API	Application Program Interface
CDC	Common Data Class
DC	Data Class
DER	Distributed Energy Resources
EVU	Energieversorgungsunternehmen
FB	Funktionsbaustein
FD	Firmwaredienste
HMI	Human Machine Interface
IEC	International Engineering Commission
IP	Internet Protocoll
KWK	Kraftwerk
LD	Logical Device
LN	Logical Node
MMS	Manufactoring Message Specification
NAR	Netzanschlussregeln
OPC	OLE for Process Control
OSI	Open Systems Interconnection
PDD	Process Data Directory
POE	Programmorganisationseinheit
RFC	Remote Field Controller
SCSM	Specific Communication Service Mapping
SNTP	Simple Network Time Protocoll
SOAP	Simple Object Access Protocol
ST	Strukturierter Text

TCP	Transmission Control Protokoll
UML	Unified Modeling Language
VK	Virtuelles Kraftwerk
WEA	Windenergieanlage
WF	Windfarm
WFM	Windfarmmanagement
WP	Windpark
WPM	Windparkmanagement
WT	Windturbine
XML	Extensible Markup Language

1 Einleitung

Einige energiepolitische Themen haben im Laufe der vergangenen Jahre rasch an Bedeutung gewonnen. Der rasant steigende Ölpreis, die breite Ablehnung gegenüber der Energiegewinnung aus Kernkraft sowie fossilen Energieträgern und nicht zuletzt der Klimawandel, rücken Fragen wie dem Ersatz der Kraftwerke nach vorn, die ihr Betriebsende in der nahen Zukunft erreicht haben. Neben Möglichkeiten der effizienteren Energieverwendung und -managements, wie der bekannten baldigen Abschaffung der Glühlampen in Australien und Europa, erbringen erneuerbare Energieträger in zunehmendem Maße ihren Beitrag zur Sicherung der Energieversorgung. Dabei kommen vorrangig Energieträger zum Einsatz, die auch unter wirtschaftlichen Aspekten interessant sind und nicht unbedingt nur der Notwendigkeit halber realisiert werden. Neben der Energiegewinnung aus Biogas, Wasser, Geothermie oder Solartechnik gehört heute zuallererst die Windenergie dazu. Die Energiegewinnung aus Windenergie gehört zu den erneuerbaren Energieträgern, die unter wirtschaftlichen Gesichtspunkten bestehen können und nicht länger ein „Zuschussgeschäft“ sind.

Windkraftpioniere bezeichnen die achtziger Jahre des 20. Jahrhunderts in der Windbranche als Beginn dessen, was heute im industriellen Maßstab produziert wird. Seit dem unterliegt die Windbranche einer stetigen Weiterentwicklung in allen beteiligten Bereichen. Der Wandel von handwerklichen Methoden zur Herstellung einzelner Windenergieanlagen bis hin zu der Umsetzung üblicher Industriestandards umfasst die gesamte Anlagentechnik. Für den Bezug auf das Thema dieser Arbeit wird zwei Merkmalen besondere Aufmerksamkeit geschenkt.

So sind mit der Steigerung der Anlagennennleistungen auch umfassende Erweiterungen der Automationstechnik mit einem höheren Integrationsgrad notwendig gewesen. Dies hat selbstredend die Auffächerung der bewegten Informationsmenge verursacht, die einen großen Anteil „windenergietypische“ Daten enthält. Zusätzlich besteht die heutige Endkundenklientel nicht länger ausschließlich aus Kleinbetreibern einzelner oder weniger Anlagen, sondern rekrutieren sich vornehmlich aus Unternehmen der Energieversorgungsbranche. So stehen die Anlagenherstellern einer Reihe von Forderungen gegenüber, die sich teilweise gegenseitig entgegen stehen. Sie

vertreten neben den eigenen Interessen auch die ihrer Energiebezugskunden oder nehmen Einfluss auf das Windparkmanagement der Betreiber (Abbildung 1.1).

Diese Arbeit wird die informationstechnische Anbindung von Windenergieanlagen gemäß der IEC 61400-25 „Communications for monitoring and control of wind power plants“ zum Kernthema machen. Nachfolgende Abschnitte dieses Kapitel geben einen geeigneten Einstieg für die folgende Ausführung der praktischen Lösung.

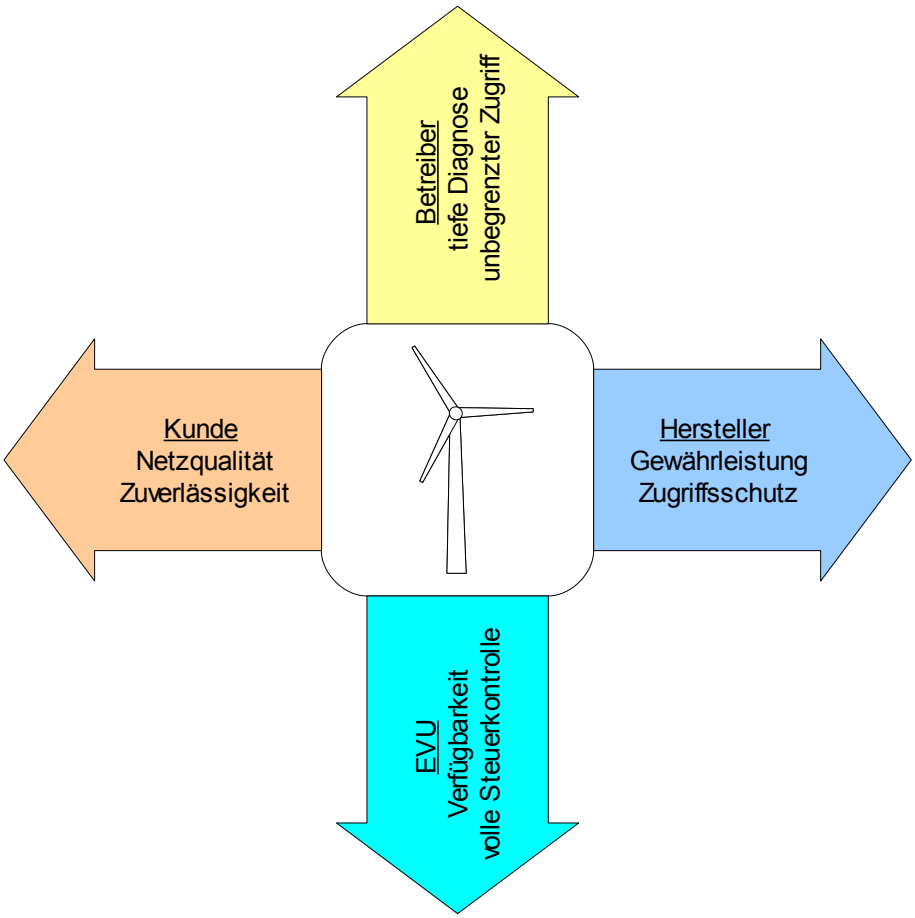

Abbildung 1.1: Konträre Interessen beim Betrieb von Windparks

2 Vorbetrachtungen

2.1 Windenergieanlagen als virtuelles Kraftwerk

Wären konventionelle Kraftwerke nur eingeschränkt steuerbar, wäre eine Energieversorgung nicht in der gewohnten Qualität realisierbar. Daher gilt es die neuen Energiequellen schon aus funktionellen Gründen in die vorhandene Infrastruktur einzubinden. Aufgrund des wachsenden Anteils am Energiemix wird dieser Anspruch auch an die erneuerbaren Energieträger gestellt. Es ist dabei gleich, welche der alternativen Energiequellen genutzt werden. Zusätzlich berücksichtigen sind die Eigenschaften der bisherigen und neuen Energiequellen, die nicht direkt vergleichbar sind. Beispielsweise ist der tageszeitbedingte Energiebedarf nicht mit dem Windangebot kohärent. Deshalb sind Windenergieanlagen nicht unbedingt als feste Größe für die Deckung des Energiebedarfs zu einem vorbestimmten Zeitpunktes geeignet. Notwendigerweise ist daher Ausgleichsenergie vor zuhalten, um o.g. Netzversorgungsqualität sicherzustellen. Dies muss jedoch nicht zwingend mit fossilen Energieträgern erfolgen. Der räumlich verteilte Bezug von Energie aus Windkraft oder eine Kombination mit anderen schon genannten, erneuerbaren Energieträgern verringert oder beseitigt das Problem, wenn diese im Verbund betrieben werden. Generell kann hier fest gestellt werden, dass die Verteilung kleiner regenerativer Kraftwerke über große Flächen eine grundlegende Eigenschaft erneuerbarer Energien ist. Um die Integration im vorhandenen Energieverbundnetz zu erreichen, ist eine geeignete Schnittstelle zu bestehenden informationstechnischen Netzwerken der EVU Voraussetzung.

An dieser Stelle wird für die Organisation von kleinen dezentralen Kraftwerken (auch: Mikro-KWK) im Verbund der Begriff des virtuellen Kraftwerkes eingeführt. Dabei kann es sich genauer um eine Organisationseinheit von gleichartigen Energieträgern handeln oder um eine gemischte Gruppe solcher (Abbildung 2.1). Räumliche Nähe ist dafür ebenso wenig Bedingung wie die Herkunft der Mikro-KWK von einem Hersteller allein.

Außer den bereits genannten Zielen der Deckung des Energiebedarfs, gibt es weitere interessante

Faktoren, die eine funktionelle Schnittstelle mit virtuellen Kraftwerken wünschenswert machen. Im Zusammenhang mit der Steuerung der Energie zu Grund- und Spitzenlast können sonst kostenintensive Minutenreserven aus mindestens einem virtuellen Kraftwerk gebildet werden. Weiterhin besteht die Möglichkeit konstruktiv verschiedenartige Ausführung zu nutzen, indem beispielsweise Einspeisungen kapazitiver oder induktiver Blindleistung (Phasenschieberbetrieb) beigetragen werden. Nicht zuletzt soll erwähnt werden, dass es im Einzelfall auch möglich wäre einen Netzaufbau nach eingetretenem Totalausfall („Blackout“) durch Lieferung der notwendigen Blindleistung für größere Kraftwerke mit Asynchrongeneratoren zu initiieren.

2.2 Eine Schnittstelle für virtuelle Kraftwerke

Zusammenfassend lassen sich aus dem letzten Abschnitt genannten Fakten Forderungen ableiten, die an eine Schnittstelle zu richten sind.

Als erstes ist die Steuerung der Energiequelle zu benennen. Dies beinhaltet das Starten, Stoppen des virtuellen Kraftwerkes sowie den Einfluss auf die Höhe der eingespeisten Energie und alle dafür notwendigen Informationen. Es handelt sich dabei um nichts mehr als den minimalsten Funktionalitätsumfang. Die zweite Forderung ist eine logische Erweiterung der vorgenannten. Sind zusätzliche Services möglich und erwünscht, sind diese zusätzlichen Funktionen zu implementieren. Sie greifen weiter in das System ein. Beispielhaft sei die Steuerung des Erregerstromes an einer Synchronmaschiene genannt werden, um einen Phasenschieberbetrieb zur Lieferung kapazitiver oder induktiver Blindleistung umzusetzen.

Die dritte Forderung setzt weitere Datenquellen zur Zustands- und Verfügbarkeitsüberwachung voraus, die Auskunft über Betriebszustände bzw. den Energieträger geben. Je nach konstruktiver Ausführung sind beispielsweise Informationen zu Rotorblattwinkel und Turmvibrationen bzw. Wind- und Temperaturverhältnissen einer Windenergieanlage verfügbar. Grundlegende Eigenschaft dieser Datenquellen ist die Existenz von (theoretisch endlosen) Datenströmen. Im Falle der Zustandsüberwachung sind Grenzwerte von Belang, deren Überschreitung Schaden oder Verlust bedeuten können. Verfügbarkeitswerte sagen aus, in welchen Maße der Energieträger nutzbar ist, d.h. wieviel Energie aufgrund der aktuellen Windgeschwindigkeit zu erwarten ist. Um die drei bisher genannten Forderungen umzusetzen, ist das Einbringen einer vierten Forderung notwendig. Da virtuelle Kraftwerke, wie schon erwähnt, aus Mikro-KWK unterschiedlicher Hersteller bestehen können, sind Maßnahmen zur Vereinheitlichung der Schnittstelle notwendig. Die Erfüllung der Forderungen mit mehreren herstellerabhängigen Schnittstellen wäre nur mit unverhältnismäßig

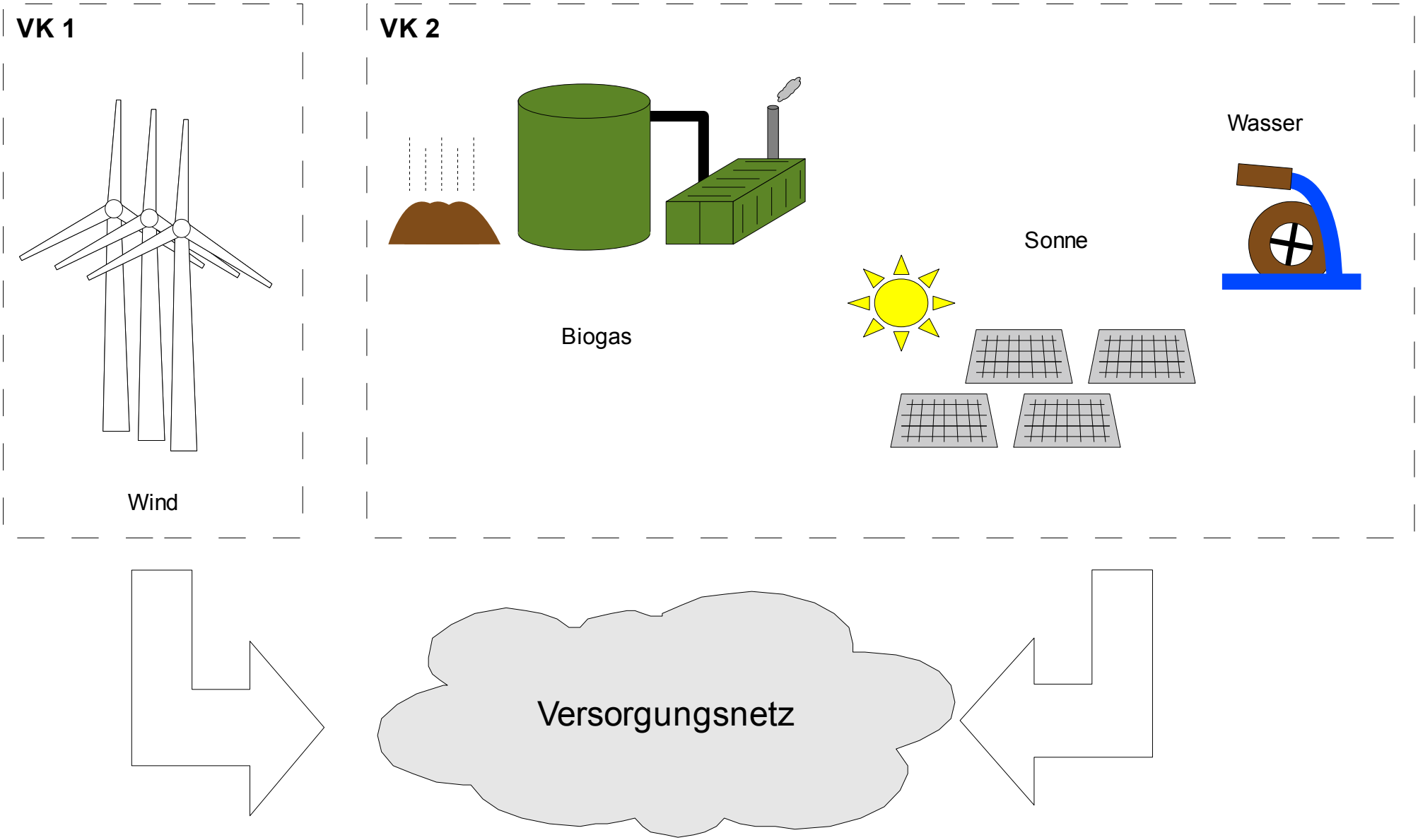

Abbildung 2.1: Ein virtuelles Kraftwerk nutzt eine oder verschiedenartige Energieträger

hohem Adaptionsaufwand zu realisieren.

2.3 Aufbau einer Windenergieanlage

Diese Arbeit wird sich mit dem Fall beschäftigen, in dem ein virtuelles Kraftwerk nur aus Windenergieanlagen, also einem Windpark, besteht. Um den Zusammenhang zu den im vorhergehenden Abschnitt vorgebrachten Forderungen an eine Schnittstelle zu einem solchen virtuellen Kraftwerk herstellen zu können, wird der Aufbau einer Windenergieanlage im folgenden erläutert. Es wird später von Belang sein, mit welchen Daten von den Komponenten zu rechnen ist.

Windenergieanlagen heutiger Zeit finden ihre Vorläufer bereits ist bekannten Bauwerken, z.B. aus dem Müllergewerbe, die hunderte Jahre alt sind. Im zwanzigsten Jahrhundert kamen neue mit dem Stand der Technik neue Methoden der Windenergienutzung hinzu, z.B. senkrecht arbeitende Darrieus-oder Savonius-Rotoren bzw. das Vorhandensein leistungselektronischer Bauelemente. Hier sei erwähnt, dass nicht alle Windenergieanlagenhersteller ein identisches Produkt hervorbringen. Der Markt wird jedoch von Firmen geprägt, deren wesentliche Eigenschaften einen Vergleich in „Form und Farbe“ zulassen. Einen schematischen Aufbau zeigt Abbildung 2.2.

Um die kinetische Energie des Windes (genauer: Geschwindigkeit der Gasteilchen, aus denen sich die Erdatmosphäre am jeweiligen Nutzungsort zusammensetzt) für die Gewinnung elektrischer Energie zu nutzen, bedarf es einer Umwandlung von der mechanischen in elektrische Energie. Die Aufnahme der mechanischen Energie geschieht mittels *Rotorblätter*, die im gleichen Winkel zueinander mit verzahnten Drehverbindern an eine *Nabe* montiert sind. Über ein in die Verzahnung eingreifendes Antriebssystem, das *Pitchsystem*, kann Einfluss auf den Betrag der aufgenommenen Energie genommen werden, in dem der Anstellwinkel zum Wind verändert wird. Von der Nabe aufgenommene Rotationsenergie leitet die *Hauptwelle* an ein *Getriebe*. Oft ist ein *Hauptlager* Teil dieser Transmission. Neuere Konstruktionen bevorzugen z.T. die Integration der Hauptwelle in das Getriebe. Die Hauptwelle erreicht in allgemeinen nur geringe Drehzahlen (<20 U/min, deshalb auch „langsame Welle“), überträgt jedoch sehr große Drehmomente. Auf der Abtriebsseite des Getriebes verfügbare Rotationsenergie wird mit deutlich höheren Drehzahlen (>1500 U/min, “schnelle Welle“) über ein Kupplung an einen *Generator* geleitet. Auf der „schnellen Welle“ ist zudem eine *Scheibenbremse* angeordnet. Im Generator findet die Umwandlung in mehrheitlich elektrische Energie statt, die über leistungselektronische Stellglieder, wie Voll- oder Teil*umrichter* und Schalttechnik zur Netzankopplung über Leitungen im *Turm* zum Netzübergabepunkt geleitet werden. Überlicherweise wird der Netzübergabepunkt mit der Einrichtung einer Substation

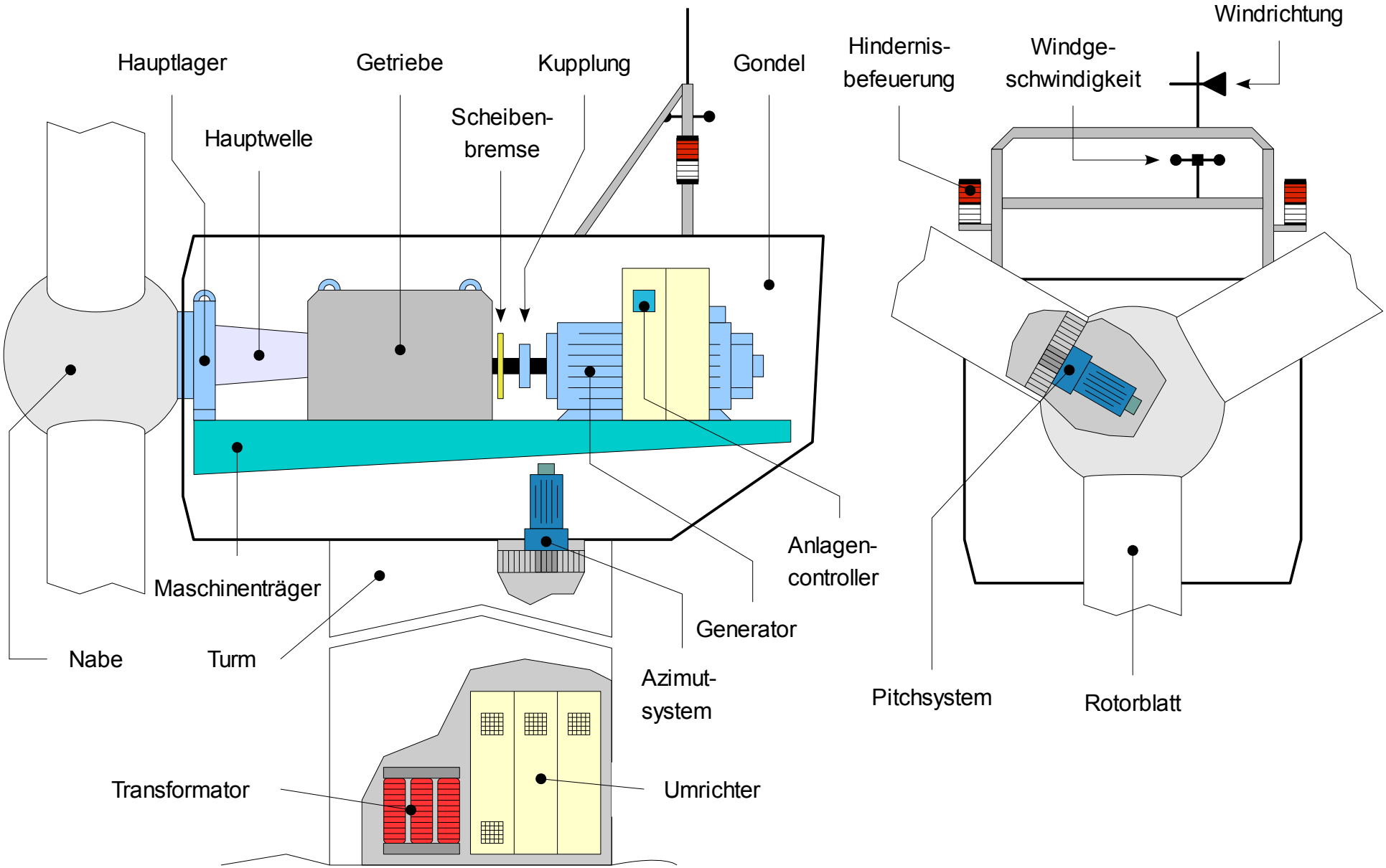

Abbildung 2.2: Schematischer Aufbau einer Windenergieanlage

realisiert. Innerhalb des Turmes befindet sich zur Anpassung an ein anderes Spannungsniveau ein *Transformator*. Dieser vereint zumeist auch die Funktion des Eigenbedarfs zur Deckung des Energiebedarfs der Systeme einer WEA in sich.

Um die Energieumwandlung in elektrische Energie durchzuführen, sind zusätzliche konstruktive Elemente notwendig. So ruht der gesamte Antriebsstrang nebst Hilfsaggregaten auf einer *Maschinenträger*. Er ist über einen verzahnten Drehverbinder mit dem Turm verbunden. Ein Antriebssystem, das *Azimutsystem* (Yawsystem), greift in die Verzahnung ein und ermöglicht die Nachführung der *Gondel* zur Windrichtung. Dafür erforderliche Führungsgrößen werden aus der Windrichtung und -geschwindigkeit gewonnen. Für diese Messeinrichtungen existiert auf dem Gondeldach ein so genannter Wettermast, auf dem auch eine Hindernisbefeuerung für den Flugverkehr montiert ist. Optional davon existiert häufig ein Messmast mit Messgeräten für meteorologische Zwecke (nicht dargestellt). Um die Anlage in ihrer Gesamtheit jedoch überhaupt steuern und regeln zu können, ist ein *Anlagencontroller* nötig. Er stellt im Zusammenhang mit der IEC 61400-25 eine wesentlich Informationsquelle dar.

Damit sind alle wesentlichen Komponenten beschrieben, deren Kenntnis für den weiteren Zusammenhang notwendig sind. Der Autor verzichtet auf weiterführende Erklärungen zu Konstruktionsmerkmalen, Hintergründen und Zweck aller Komponenten, da der Fokus dieser Arbeit dem Informationsaustausch und der Schnittstellengestaltung gilt [HEI03][GAS07].

2.4 Beschreibung des Informationsaustausches

Um die Argumentationslinie dieser Einleitung richtig fortzuführen, ist ein Blick auf die zu erwartenden Datenquellen angemessen. Windenergieanlagen sind heute überwiegend mit einem Bussystem ausgestattet. Solche ohne ein Bussystem führen alle digitalen uns analogen Signale auf einen zentralen Betriebsführungsrechner bzw. Anlagencontroller. Beiden gemeinsam ist die Tatsache, dass damit (fast) alle Datenströme über den Anlagencontroller fließen. Er bildet auf Seiten der WEA den Kommunikationspartner zur Aussenwelt. Diese Daten können uni- oder bidirektional zwischen der WEA und dem Kommunikationspartner übermittelt werden.

Es sei an dieser Stelle bemerkt, dass einige elektronische Geräte auch außerhalb des Bussystems bzw. des Betriebsführungsrechners Kommunikationsschnittstellen aufweisen. Die ist im normalen Betriebsablauf einer WEA jedoch eher nicht die Regel und kommt z.B. bei Service- oder Inbetriebnahmearbeiten zum Einsatz. Zudem weichen die auf solchen Wegen angebotenen Kommunikationsdienste nicht sehr von denen einer Busschnittstelle ab.

Da also alle durch den Anlagencontroller für die Aussenwelt verfügbaren Daten durch die Komponenten der WEA geliefert werden, ist eine Betrachtung des Informations- und Datenaufkommens von Belang. Die Hauptbestandteile einer Windenergieanlage wurden bereits benannt. Hier sollen einige von ihnen beispielhaft näher aus der Sicht des Datenverkehrs beschrieben werden.

Als sinnhaftes Bauteil einer WEA trifft man überall auf einen Generator. Wahlweise sind Asynchron- oder Synchronmaschinen im Einsatz. Neben den Ertragsgrößen der elektrischen Energie sind Zustandswerte der notwendigen Hilfseinrichtungen wichtig. Folgende Messwerte können also vom Generator erwartet werden:

- Spannungen, Ströme aller drei Phasen (analog),
- Temperatur des Kühlsystems (analog) und
- Zustände der Überspannungselemente und des Bürstenverschleißes (digital).

Der Generator gehört zu den Komponenten, die üblicherweise keine Einrichtung zur Übermittlung von Befehlen irgendeiner Art besitzen.

Ganz ähnlich verhält es sich mit dem häufig vorhandenen Element zur Drehzahlanpassung des Rotors für die Generatorwelle, dem Getriebe. Auch dort fallen nur Sensorwerte der Hilfssysteme an, namentlich...

- des Betriebszustandes der Ölumlaufpumpe (digital),
- der Temperatur des Ölsumpfes (analog) und
- der Temperatur des Kühlsystems (analog).

Neben weiteren Baugruppen, die alleinig durch einseitig übermittelte Sensorwerte in der Menge der Daten auftauchen, existieren jedoch auch weit komplexere Komponenten in einer Windenergieanlage. Sie benötigen eine weitergehende, bidirektionale Kommunikation, um ihre Funktion zu erfüllen. Außer der Rückmeldung aktueller Werte, z.B. Ist-Werte, sind Kommandos und Parameter zur Komponente zu übertragen. Da sie überwiegend Teil eines Regelkreises sind, hat der Informationsfluss echtzeitlichen Ansprüchen genügen. Typische Baugruppen dieser Art sind das Pitchsystem zur Blattverstellung oder ein Hauptumrichter bei doppelt gespeisten Asynchronmaschinen. Im Falle des Pitchsystems werden beispielsweise Blattwinkelwerte zeitäquidistant als Sollwerte übertragen und Ist-Werte als Teil einer Regelung ebenso zurückgesandt. Neben der Kommunikation dieser Art existiert jedoch auch ein unregelmässiger und

nicht zeitkritischer Informationsaustausch, der für Konfigurations- und Zustandswerte genutzt wird. Es gibt also es eine Unterscheidung bzgl. Zeitrahmen und Zeitpunkt sowie dem funktionellen Inhalt der Übertragung. Daten zur Steuerung der WEA treten fast ausnahmslos zeitlich unregelmäßig auf. Sie werden manuell oder automatisch veranlasst. Daneben existieren theoretisch endlose, überwiegend von der WEA gesandte, Datenströme. Wann Daten unregelmäßig oder als Datenstrom auftreten, hängt vom Zweck ab. In einem angenommenen Fall kann die dauerhafte Überwachung von elektrischen Kennwerten, wie Spannungen, Strömen, Leistungen oder die ständige Rückmeldung eines Ist-Wertes in einem Datenstrom von Bedeutung sein. Dem gegenüber ist bei einer Steuer- oder Parametrierungsanweisung aber von einer zeitlich unregelmäßigen Datenübertragung auszugehen.

Neben dem zeitlichen Auftreten spielt die Frage nach dem Datentyp eine Rolle bzw. dessen, was die Daten repräsentieren. Mit Integration in ein Bussystem bzw. in die Anlagensteuerung werden alle Werte digitalisiert. Das trifft für Analogwerte ebenso zu wie für schon digital verfügabre Informationen. In Abhängigkeit vom Inhalt der Informationen sind Datentypen, Wertebereich, Genauigkeit oder Auflösung und ggf. eine Einheit oder ein Faktor zuzuordnen.

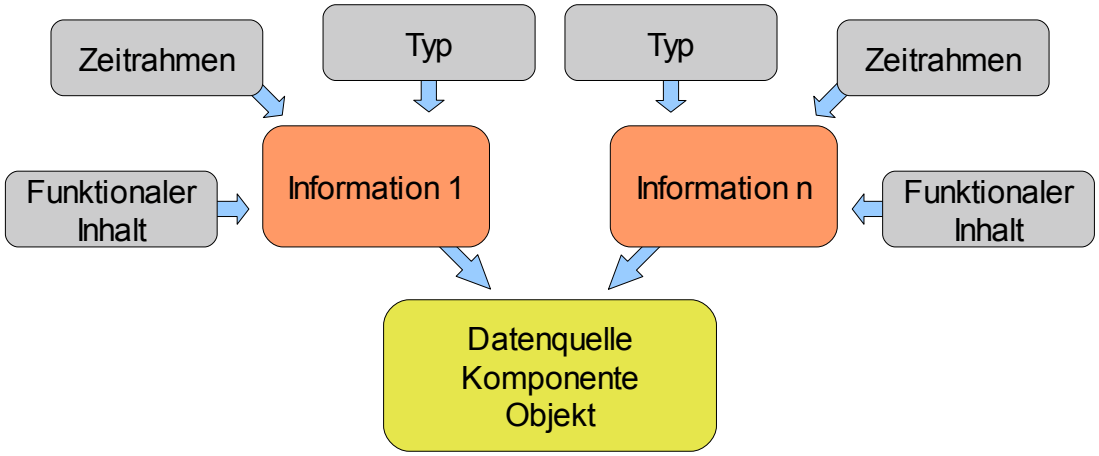

Abbildung 2.3: Informationen mit ihren Eigenschaften lassen sich Objekten bzw. Datenquellenzuordnen

Letztlich sind die Daten in typischen Baugruppen einer WEA zuzuordnen, die sich in sehr vielen Windenergieanlagen wiederfinden. Der Ansatz der Objektorientierung liegt hier schon auf der Hand, da sich die Objekte mit den vorhandenen Datenfeldern als Eigenschaft bzw. Attribut beschreiben lassen. Die Zuweisung dieser Informationen zu Gruppen oder Objekten bringt neben anderen Vorteilen die der Übersichtlichkeit und Modularisierung mit (Abbildung 2.3). Auf diese Thematik wird in den nachfolgenden Abschnitten tiefer eingegangen.

3 Analyse der IEC 61400-25

3.1 Überblick und Einordnung der Normengruppe

Das International Electrotechnical Committee (IEC) stellt maßgebliche Standards für den Bereich der Elektrotechnik bereit. Das IEC wurde 1906 in London gegründet. Es bildet die Dachorganisation für die jeweiligen nationalen Einrichtungen. In der Bundesrepublik Deutschland wird diese durch Aufgabe durch das Deutsche Institut für Normung e.V. (DIN) wahrgenommen. Konkrete Vorschläge und Ausarbeitungen werden durch Technical Committees (TC) vorgelegt. Im Falle der Normengruppe IEC 61400, es sind nur Normen für die Windenergietechnik enthalten, geschieht das durch das TC88 „Windturbines“. Sie widmet sich ausschließlich Themen aus der Windenergiebereich. Bei der IEC 61400-25 handelt es sich um eine Spezialisierung der IEC 61850 „Communication networks and systems in substations“, die eine objektorientierte Sicht der Informationen in einer Substation vorsieht und nicht Teil der genannten Normengruppe ist. Das bedeutet an dieser Stelle den Ausschluss einer Substation aus der Betrachtung dieser Arbeit, während die angeschlossenen Windenergieanlagen sehr wohl im Zentrum dieser stehen.

Tabelle 3.1 listet alle Normen der IEC 61400 Gruppe auf, in deren Kontext die IEC 61400-25 „Communications for monitoring and control of wind power plants“ steht. Diese Norm setzt sich aus sechs Teilen zusammen. Für die weitere Betrachtung sind jedoch nur die Teile eins bis fünf von Belang, die im Zeitraum von Dezember 2006 bis August 2008 in der ersten Fassung veröffentlicht wurden. Der Teil 6 befindet sich hingegen erst im frühen Entwurfsstatus und befasst sich thematisch mit dem für eine WEA nicht zwingend betriebsnotwendigen Condition Monitoring System, das in jüngster Zeit aufgrund von Forderungen der Versicherer, vornehmlich zur Systemüberwachung von Hauptgetrieben genutzt wird.

Für die Umsetzung der IEC 61400-25 sind die Teile 2, 3 und 4 wichtig. Sie enthalten die Beschreibungen des Kommunikationsmodells in Abbildung 3.1 , welches aus Daten- und -austauschmodell und Mapping besteht. Nachfolgend wird auf diese Teile näher eingegangen.

IEC	***Titel***
IEC 61400-1	Design requirements
IEC 61400-2	Design requirements for small wind turbines
IEC 61400-11	Acoustic noise measurement techniques
IEC 61400-12-1	Power performance measurements of electricity producing wind turbines
IEC 61400-12-2	Power performance of electricity producing wind turbines based on nacelle anemometry
IEC 61400-12-3	Wind farm power performance testing
IEC 61400-13	Measurement of mechanical loads
IEC 61400-14	Declaration of apparent sound power level and tonality values
IEC 61400-21	Measurement and assessment of power quality characteristics of grid connected wind turbines
IEC 61400-22	Conformity testing and certification of wind turbines
IEC 61400-23	Full-scale structural testing of rotor blades
IEC 61400-24	Lightning protection
IEC 61400-25-1	Communications for monitoring and control of wind power plants Overall description of principles and models
IEC 61400-25-2	Communications for monitoring and control of wind power plants Information models
IEC 61400-25-3	Communications for monitoring and control of wind power plants Information exchange models
IEC 61400-25-4	Communications for monitoring and control of wind power plants Mapping to communication profile
IEC 61400-25-5	Communications for monitoring and control of wind power plants Conformance testing
IEC 61400-25-6	Communications for monitoring and control of wind power plants Logical node classes and data classes for condition monitoring
IEC 61400-26	Availability for wind turbines and wind turbine plants

Tabelle 3.1: IEC 61400 Normengruppe bezieht sich nur auf Windenergieanlagen [IECWWW]

3.2 Die Übersicht in IEC 61400-25-1

Die IEC 61400-25-1 steht als erste Teil für die Einführung und Übersicht für alle weiteren Teile. Es werden weiterhin wichtige Referenzen benannt, allem voran die betreffenden Teile der IEC 61850. In Abbildung 3.1 findet sich die Darstellung des maßgeblichen Kommunikationsmodells. Das Kommunikationsmodell sieht die informationstechnische Anbindung von einzelnen oder mehreren Windenergieanlagen bzw. eines Windparks mit externen Kommunikationspartnern vor. Es bedient sich dabei einer Client-Server Systemarchitektur. Bei Einem Server handelt es sich um einen Teilnehmer eines Netzwerkes, der als Diensteanbieter für einen oder mehrere Clients als Dienstenutzer zur Verfügung steht [BRA07]. Es sei darauf hingewiesen, dass die Begriffe Client und Server je für ein Client – bzw. Serverprogramm stehen und die Hardware explizit bezeichnet wird.

Wesentliche Bestandteile sind das Datenmodell (IEC 61400-25-2) und das Datenaustauschmodell (IEC 61400-25-3). Alle Modelle werden in nachfolgenden Abschnitten näher erklärt. Kommunikation findet über eines von fünf möglichen Kommunikationsprofilen (IEC 61400-25-4) statt. Um objektorientierten Hintergrund dieser Bestandteile zu verstehen, ist zuerst ein Blick auf den Sinn der Objektorientierung und ihrer Sichtweisen ratsam.

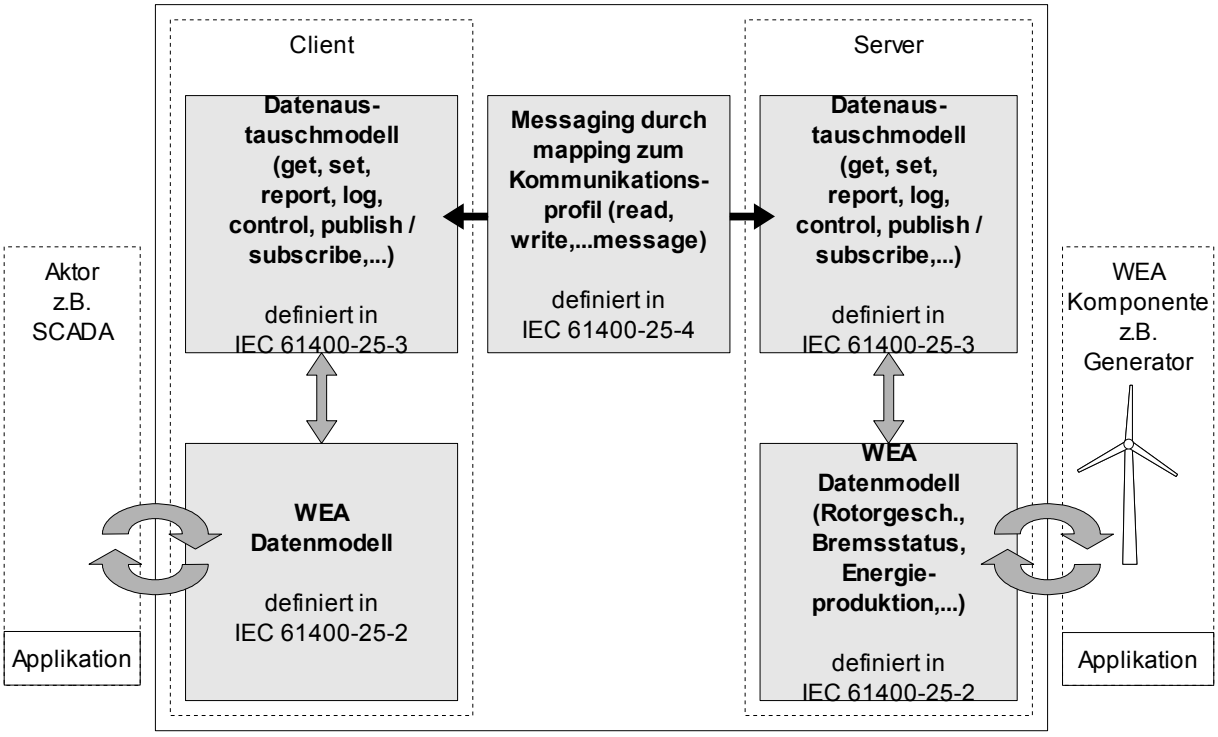

Abbildung 3.1: Konzeptionelles Kommunikationsmodell nach IEC 61400-25

3.3 Merkmale der Objektorientierung

Die Objektorientierung hat ihre Wurzeln in den sechziger Jahren des vergangenen Jahrhunderts. Die einst sehr vorherrschenden prozeduralen und strukturierten Programmiersprachen und Ansätze boten keine Lösungen für neue auftauchende, komplexere Aufgabenstellungen. Typische Eigenschaften der prozeduralen Programmierung waren u.a. die fehlende Einheit von Daten und Funktionen (Abbildung 3.2). Da sich Fehler, wie unkorrekte Zugriffe auf Daten oder eine unbeabsichtigte Verwendung nicht initialisierter Daten nicht ausschließen lassen, ist diesem Ansatz eine geringere Zuverlässigkeit zuzuschreiben. Der Effekt kann insbesondere auftreten, wenn Änderungen, wie z.B. Wartungsarbeiten, nachträglich am Code vorgenommen wurden.

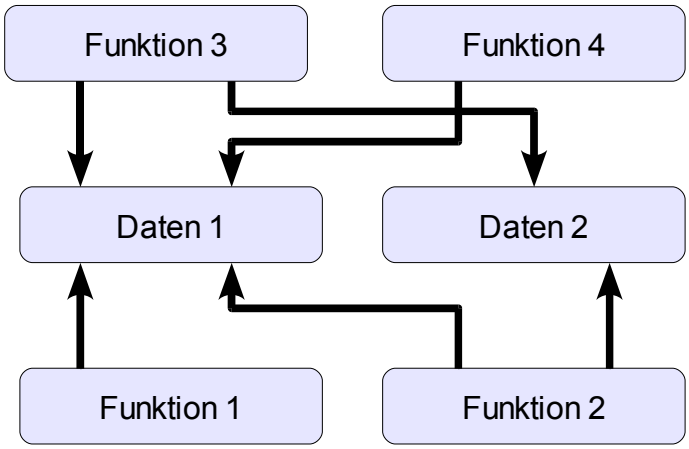

Abbildung 3.2: Prozeduraler Zugriff auf Daten durch Funktionen

Wesentlich mehr Erfolg konnte ein Ansatz verbuchen, der Daten (Eigenschaften) und Funktionen (Methoden) in einer Einheit zusammenführt. Besonders in Bezug auf die IEC 61400-25 sind die wesentlichen vier Merkmale der Objektorientierung zu benennen.

Unter dem Begriff der **Datenabstraktion** versteht man die Möglichkeit, abstrakte Datentypen (Klassen) zu definieren. Die so definierte Klasse beschreiben Objekte mit ihren Eigenschaften und Fähigkeiten. Von diesem Mechanismus wird in der IEC 61400-25 im ganz wesentlichen Gebrauch gemacht und wäre ohne dies gar nicht zu verwenden. Man kann einer Windenergieanlage allgemeine Eigenschaften zugestehen, wie Leistung, Turmhöhe, Rotordurchmesser oder Anzahl der Generatoren und ebenso allgemeine Fähigkeiten, wie starten oder stoppen.

Ausgewählte Elemente eines Objektes können vor einem Zugriff von außen geschützt werden. Die **Datenkapselung** sieht nur eine „öffentliche“ Schnittstelle zu anderen Objekten vor, der versehentliche Zugriff auf Daten ist verhindert. Die innere Struktur der Klasse bzw. des Objekts

bleibt dadurch nach außen hin verborgen. Der Datenzugriff und ist nur mittelbar kontrolliert durch die „öffentliche Schnittstelle“ im allgemeinen mittels Methoden erreichbar. In dieser Weise verwalten sich Objekte also selbst.

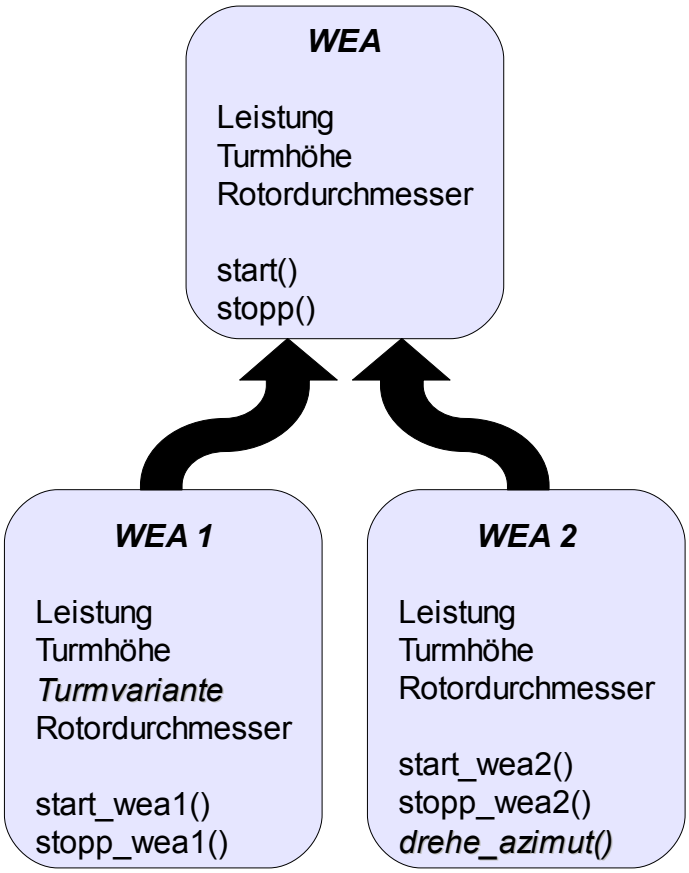

Abbildung 3.3: Die Klassen WEA 1 und 2 erben von Basisklasse WEA

Durch die Möglichkeit der **Vererbung** wird neuen Objekten bei der Erstellung ein bereits bestehendes Objekt (Basisklasse) vorgegeben, deren Fähigkeiten und Eigenschaften das neue Objekt erhält. Letzteres kann diese Fähigkeiten und Eigenschaften jedoch ergänzen oder neu definieren. In Abbildung 3.3 ist dieses Vorgehen dargestellt. Ein Objekt wird durch die Klasse *WEA* verkörpert. Die Klassen *WEA 1* und *WEA 2* sind von der Klasse *WEA* abgeleitet und erben so alle Eigenschaften und Funktionen. Jedoch ergänzen sie die Basisklasse um eine Eigenschaft *Turmvariante* bzw. eine Funktion *drehe_azimut()*, die sie zusätzlich den geerbten Eigenschaften und Funktionen nutzen können. Es bestünde auch die Möglichkeit bereits geerbte Eigenschaften und Funktionen zu überlagern. Zu diesem Zweck sind diese in der abgeleiteten Klasse neu zu erstellen. Instanzen der abgeleiteten Klassen nutzen die jeweilige überlagerte Eigenschaft oder Funktion, die den Instanzen der Basisklasse hingegen aber unbekannt sind. Man bezeichnet diese Arbeitsweise als **Polymorphie** (griech: Vielgestaltigkeit).

Die Vererbung ist zudem ein wichtiges Mittel, bereits erstellte, in sich abgeschlossene Objekte wiederzuverwenden. Für die Benutzung ist die Kenntnis des äußeren Verhaltens des Objektes wichtig, nicht die genaue innere Struktur. Es bietet sich die Möglichkeit, Sammlungen von Objekten in Bibliotheken zusammenzufassen und zu verteilen. In Bezug auf das behandelte Thema wird das aber nur geringfügig nutzbar sein.[PRI98]

3.4 Das Datenmodell nach IEC 61400-25-2

3.4.1 Überblick

Die Erstellung eines Datenmodells dient der Erfassung aller Informationen, deren Bearbeitung im weiteren vorgesehen ist. Es verkörpert den gesamten Datenbestand, der für alle Methoden zur Verfügung steht. Man spricht in diesem Zusammenhang auch von einer Hardware-Virtualisierung. Dabei bleiben die genauen konstruktiven und funktionellen Zusammenhänge der Hardware, hier die Windenergieanlage, unberücksichtigt und spielen für das Datenmodell keine Rolle. Es wird ausschließlich die Repräsentation der Daten des betrachteten Objekts für einen bestimmten Zweck beabsichtigt und nach diesen Gesichtspunkten modelliert.

Für die Modellierung stellt die IEC 61400-25-2 ein Set von Modellierungselementen bereit. Zu modellierende Informationen werden in Kategorien nach [IEC400-25-1], Table 3 eingeteilt:

1. Process information
    1. State information
    2. Analogue information
2. Control information
    1. Control information
3. Derived information
    1. Cumulative information
    2. Historical information

Der Fokus liegt einer objektorientierten Vorgehensweise. Das Datenmodell der IEC 61400-25 ist hierarchisch gegliedert. Daher macht es später durchaus Sinn, eine Baumstruktur für die Darstellung zu wählen. Es erfolgt die Erklärung der Modellierungselemente vom obersten Element aus. Die

Zusammenhänge der Modellierungselemente sind in Abbildung 3.4 dargestellt.

3.4.2 Physical Device

Das Physical Device besitzt eine besondere Stellung. Zwar hält es nicht die höchste Position, jedoch existiert es genau ein Mal und hostet per Definition alle Logical Devices. Für die spätere Betrachtung sei angemerkt, dass es in einer Baumdarstellung deshalb Sinn macht, es einmalig als ranghöchstes Element darzustellen, dem sich alle Logical Devices unterordnen. Bei der Modellierung kommt dem Physical Device des Servers oder des Clients zu.

3.4.3 Logical Device

Die höchste Ebene wird durch ein oder mehrere Logical Device(s) repräsentiert. In Bezug auf die Windenergieanlage handelt es sich dabei zumeist um eine Windenergieanlage selbst aus der Summe seiner Einzelteile. Jedes Logical Device zerfällt in mehrere Logical Nodes, um die Einzelteile im Datenmodell abzubilden.

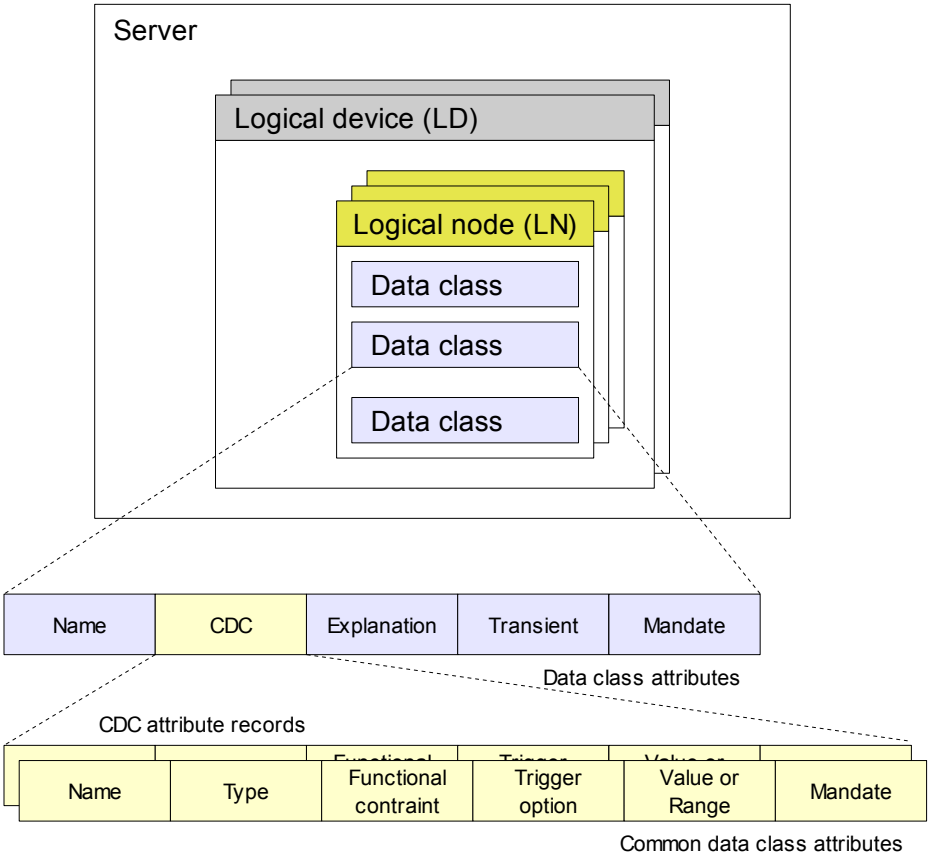

Abbildung 3.4: Struktur des Windenergieanlagen Datenmodells nach [IEC400-25-1]

3.4.4 Logical Node

Eine Modellierung über die Logical Devices hinaus ist mit vorgenannten Logical Nodes vorgesehen. Logical Nodes sind Container, die alle Informationen der Windenergieanlage enthalten. Nach Zweck eingeteilt finden sich für Windenergieanlagen spezifische Logical Node in [IEC400-25-2]. Sie orientieren sich an oft auftretenden Baugruppen und Komponenten einer Windenergienanlage, wie beispielsweise Rotor, Generator, Turm oder Umrichter. Darüber hinaus existieren auch solche, die zu allgemeinen Zwecken dienen, etwa den Alarm- oder Ereignismeldungen.

Die Benennung der Logical Nodes muss eindeutig sein und setzt sich aus vier Grossbuchstaben zusammen. Aus [IEC400-25-2] stammende Logical Node nutzen als ersten Buchstaben „W“ (...für Wind“) gefolgt von drei weiteren, den Inhalt bezeichnenden, Buchstaben (Wxxx). Es werden zusätzlich weitere kompatible Logical Nodes und Benutzungsregeln aus [IEC850-7-1] und [IEC850-7-4] angewandt.

Attribute Name	*Attr. Type*	*Explanation*	*M/O*
Data			
Common Information			
Data class name	CDC	Description and range	
Status Information			
Data class name	CDC	Description and range	
Analogue Information			
Data class name	CDC	Description and range	
Control Information			
Data class name	CDC	Description and range	

Tabelle 3.2: Generelle Tabellenstruktur eines Logical Node nach [IEC400-25-1]

Logical Nodes besitzen eine standardisierte Tabellenstruktur wie in Tabelle 3.2, die die weitere Unterteilung und Zuordnung der Daten in Data Classes ermöglicht. In Tabelle 3.3 sind die Tabellenattribute erläutert. Je nach ausgewählten Logical Node sind Data Classes optional oder Pflicht (Mandatory). In letzterem Fall sind sie immer anzuwenden.

Neben den Logical Nodes für die Beschreibung technischer Größen aus den für die Kommunikation sichtbaren Werten, gibt es zwei der besonderen Art, die explizit erwähnt werden müssen. Physical Device und Logical Device müssen mit ihrem Erscheinen eigene spezielle Logical Node (LLN0 und LPDH) implementieren, die informelle Daten zu deren Beschreibung enthalten.

Data class attribute	***Description***
Attribute Name	Name der Data Class
Attribute Type	Common Data Class die die Common Data Properties definiert. Die Common Data Classes sind in [IEC400-25-2] definiert
Explanation	Kurze Erklärung des Inhalts der Data Class
Mandate	M: Manadtory (Pflicht), O:Optional

Tabelle 3.3: Data Class Attribute der Data Class im Logical Node nach [IEC400-25-1]

Der Zusammenhang aller Logical Nodes besteht in der Verwandtschaft. Sämtliche Logical Nodes, wie LPHD, LLN0 oder systemtypische Varianten, leiten sich aus einem abstrakten Logical Node ab, der in [IEC850-7-2] definiert ist. Damit werden alle Eigenschaften und Fähigkeiten vererbt (Abbildung 3.5).

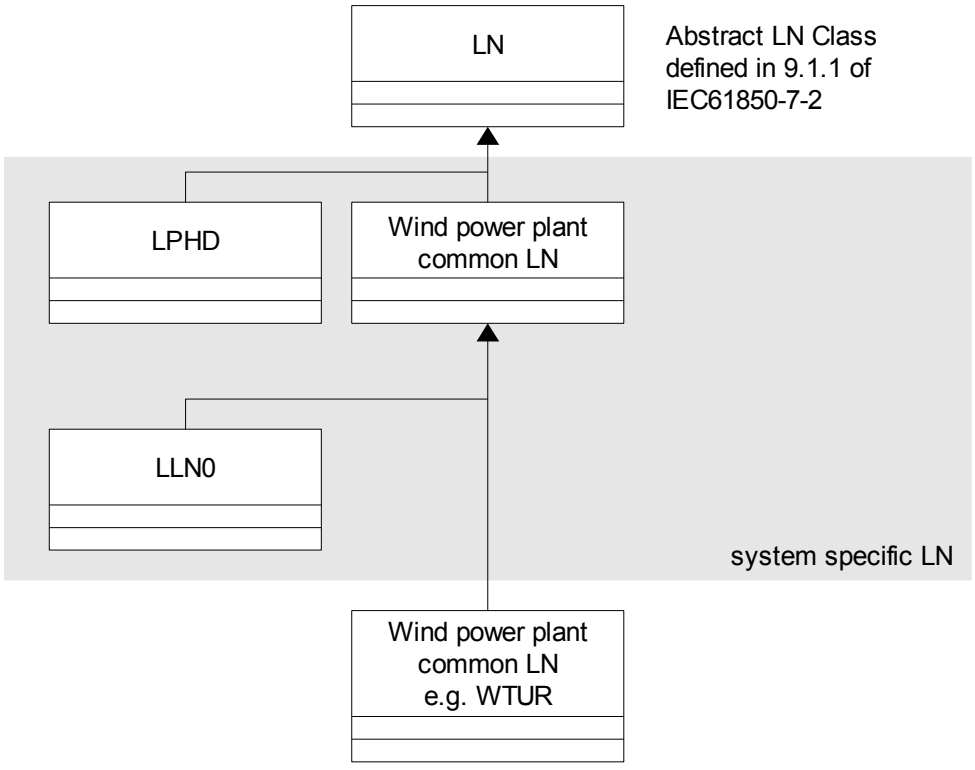

Abbildung 3.5: UML Klassendiagramm zu den Beziehungen der Logical Nodes nach [IEC400-25-2]

3.4.5 Data Class

Alle Daten der Logical Nodes werden als benannte Attribute repräsentiert, die eines einfachen (32 Bit Integer) oder komplexen Typs (komplexe Strukturvariable, aus einfachen und komplexen Typen bestehend) sein können. Es handelt sich hierbei um die sogenannten Data Classes (siehe Abbildung

3.4). Sie verweisen in einer letzten Modularisierung auf Common Data Classes.

3.4.6 Common Data Class

Data Classes erben ihre Eigenschaften von den Common Data Classes, mit denen sie assoziiert sind. Common Data Classes entsprechen dem kleinsten verfügbaren Modulierungselement. Dabei wird erneut von einer generellen Tabellenstruktur (Tabelle 3.4) Gebrauch gemacht. Die zugehörigen Erklärungen zu den Attributen sind in Tabelle 3.5 dokumentiert.

xxx class					
Attribute name	***Attribute type***	***FC***	***TrgOp***	***Explanation and Value / Range***	***M/O***
Data attribute					
Status information					
cdc attr. name	attr. type	fc		Decription and range	
Analogue information					
cdc attr. name	attr. type	fc		Decription and range	
Statistical information					
cdc attr. name	attr. type	fc		Decription and range	
Historical information					
cdc attr. name		fc		Decription and range	
cdc attr. name	attr. typeA	fc		Decription and range	
cdc attr. name	attr. typeB	fc		Decription and range	
cdc attr. name	attr. typeC	fc		Decription and range	
cdc attr. name	attr. typeD	fc		Decription and range	
Control information					
cdc attr. name	attr. type	fc		Decription and range	
Setpoint information					
cdc attr. name	attr. type	fc		Decription and range	
Description and extension					
cdc attr. name	attr. type	fc		Decription and range	

Tabelle 3.4: Generelle Tabellenstruktur einer CDC nach [IEC400-25-2]

Data class attribute	***Description***
Attribute name	Kurzbezeichnung für den Common Data Class Datensatz
Attribute type	Basis- oder zusammengesetzter Datentyp
Functional constraint	Kurze Bezeichnung für die Gruppierung der Daten (Gesamtliste in Tabelle 18, [IEC400-25-2]) Beispiele: ST Status MX Measurand CO Control SP Setpoint CF Configuration DC Description
Trigger option	Triggeroption; Umstände unter denen eine Information auftaucht, nämlich... dchg: data exchange qchg: quality change dupd: data update
Explanation/Range	Beschreibung und Bereich des Datensatzes
Mandate	M: Mandatory (Pflicht), O:Optional, siehe auch [IEC400-25-2], Table 26

Tabelle 3.5: Erklärung der Tabellenattribute der Tabelle 3.4

Verallgemeinernd kann formuliert werden, dass auf Daten in der Form

Logical_Device **.** Logical_Node **.** LN_Attribute_Name **.** CDC_Data_Group **.** CDC_Attribute_Name

zugegriffen werden kann.

Die Namenskonvention sieht für die Common Data Classes drei bezeichnende Grossbuchstaben vor.

3.4.7 Beispielszenario mit Ergänzungen zum Datenmodell

An Hand der Abbildung 3.6 wird die beispielhafte Implementation eines Datenmodells nach [IEC400-25-2] dokumentiert. Für die Darstellung wurde die bereits vorher erwähnte vorteilhafte Baumstruktur gewählt.

Das oberste Element wird vom Physical Device gebildet. Alle Daten (Datenmodell) werden in einer

RFC Steuerung gespeichert. Deshalb repräsentiert das Physical Device die RFC Steuerung. Es wird mit einem Logical Node (LPHD) modelliert. Abweichend von der üblichen Zuordnungsweise, wird LPHD nicht direkt dem RFC-Knoten unterstellt. In [IEC400-25-2] wird die Zuordnung zum Logical Device festgelegt. Aus diesem Grunde wird dieser spezielle Logical Node auch dort angeordnet. Sind weitere Logical Devices einem Physical Device zugeordnet, so ist diesen ebenfalls ein Logical Node LPHD mit dem gleichen Inhalt zuzuordnen. Alle Logical Devices innerhalb des gleichen Physical Devices haben je einen identischen Logical Node LPHD.

Die dem Physical Device (RFC) untergeordnete Elemente sind Logical Devices. In Abbildung 3.6 ist dies das Logical Device „WEA0815“. Logical Devices fassen Logical Nodes zu einer Gruppe zusammen. In diesem Fall liegt es nahe, die betreffende Windenergieanlage (hier: WEA0815) zu wählen. Allgemeine Informationen über das Logical Device sind im Logical Node LLN0 gespeichert. Auch dies ist eine Festlegung in [IEC400-25-2]. LLN0 wird auch wie LPHD einem Logical Device untergeordnet. Im Gegensatz zum LPHD sind diese jedoch je Logical Device individuell.

Neben LPHD und LLN0 gibt es beliebig viele weitere Logical Nodes, die einem Logical Device zugeordnet sind. Es handelt sich dabei um vorgegebene Logical Nodes nach [IEC400-25-2], [IEC850-7-2] und [IEC850-7-3]. Darin enthalten sind auch Regeln für die Möglichkeit der Neuerstellung von Logical Nodes. Beispielhaft soll der Datenzugriff an Hand des Logical Devices *WEA0815* gezeigt werden. Das Logical Device ist dem Physical Device, dem RFC, unterstellt, über den alle Informationen der *WEA0815* nach außen sichtbar sind.

Dem Logical Device ist in unserem Beispiel ein typisches Logical Node *WGEN* zugeordnet. Die Tabellenstruktur entspricht der allgemeinen Tabellenstruktur für Logical Nodes in Tabelle 3.2. So enthält die Tabelle von *WGEN* auch Data Classes über Statusinformationen. Das Beispiel verwendet die optionale Data Class *OpTmRs* vom Typ *TMS*. *TMS* ist eine Common Data Class. In dieser Common Data Class wird auf das Attribut *oldTmVal* Bezug genommen. Über diese Gruppe sind alle untergeordneten Informationen in der Common Data Class *INS* zu erreichen. Nach vorgenannter Definition wäre also ein Zugriff in der Form

WEA0815 . WGEN . OpTmRs . oldTmVal . stVal

möglich.

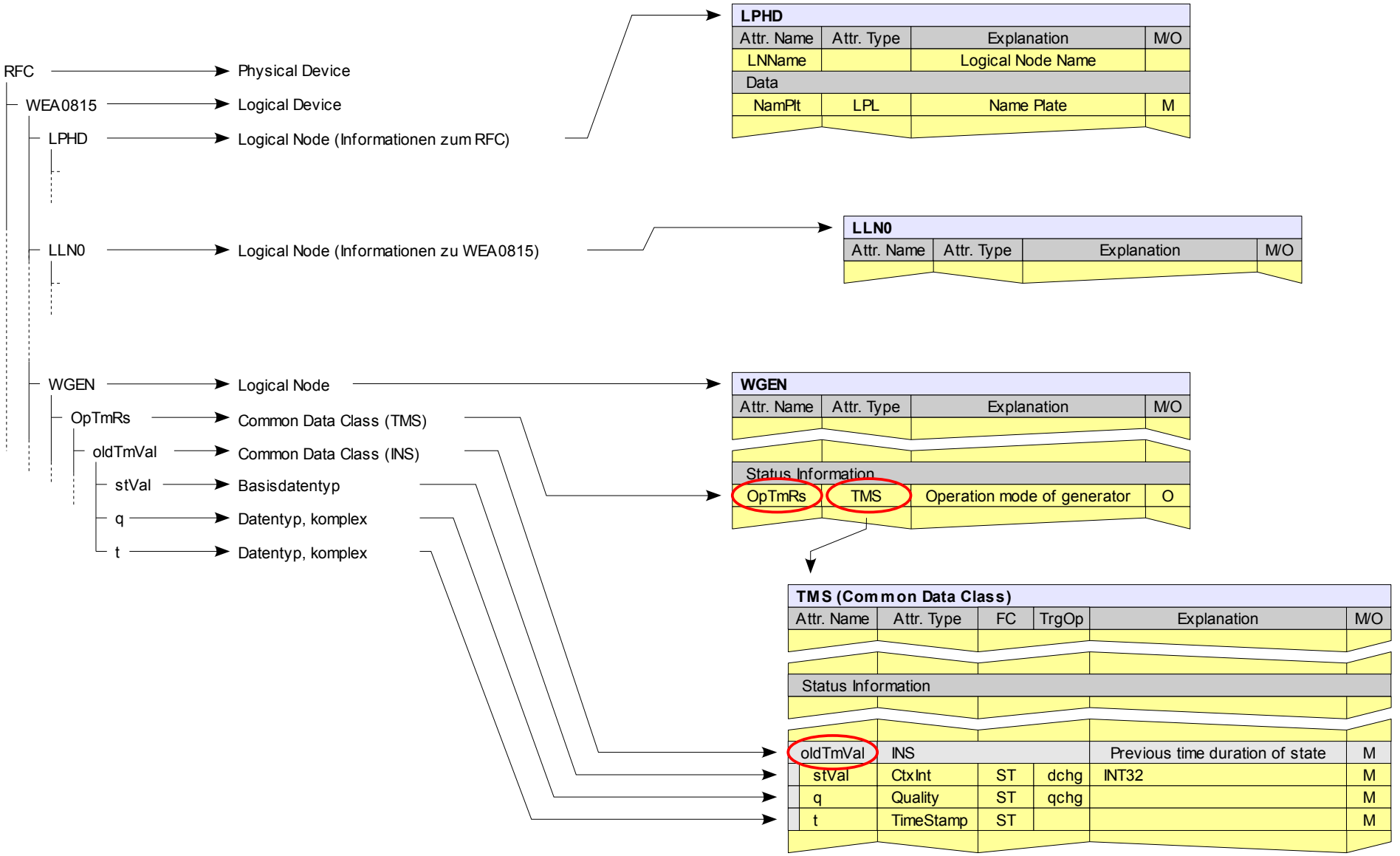

Abbildung 3.6: Darstellung der Zugriffsweise in einer Datenstruktur aus dem Informationsmodell

Neben diesem Beispiel existieren jedoch in der Anwendung komplexere Strukturen sich weiter verschachtelnder CDC. Die korrekte Objektreferenz wird aufgebaut, bis ein Attribut Name zu einem Basisdatentyp ([IEC400-25-2], S. 48, Table 27) zugeordnet worden ist. Ein exemplarisches Beispiel mit einigen Kommentaren findet sich in [IEC400-25-4] S. 27. In diesem Beispiel finden sich Common Data Classes, in denen ein Attribute Typ eine Common Data Class referenziert oder ein komplexer Datentyp zugeordnet wurde, der sich aus vorgenannten Basisdatentypen zusammensetzt.

3.5 Das Datenaustauschmodell nach IEC 61400-25

3.5.1 Überblick

Neben dem modellierten Datenaufkommen einer Windenergieanlage, ist der Datenaustausch ein zweiter elementarer Baustein des IEC 61400-25 Kommunikationsmodells (Abbildung 3.1). Die IEC 61400-25 definiert dabei nur die Serverfunktion, was im Schwerpunkt dieser Arbeit liegt. Zur Beschreibung dessen werden Informationen zu den Servicemodellen neben denen des Abstract Communication Service Interface als Schnittstelle mit ihren wesentlichen Merkmalen zur Außenwelt vermittelt.

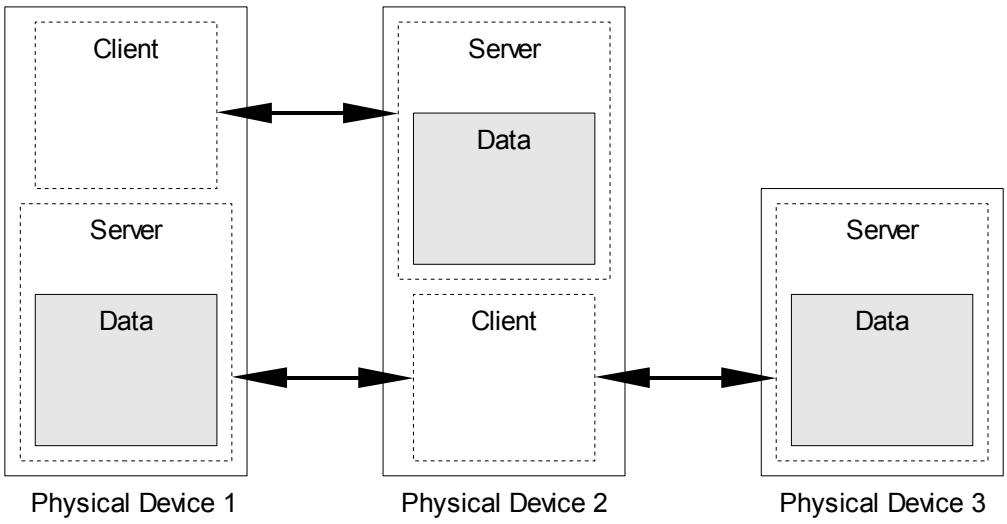

Abbildung 3.7: Server und Daten sowie Clients sind auf Physical Devices verteilt (nach [IEC400-25-1])

Im Informationsmodell repräsentiert das Physical Device den Remote Field Controller mit seiner Funktion als Server oder Proxy. Dem entsprechend ist ein Zugriff von einem oder mehrerer Clients möglich. Da die IEC 61400-25 kein zentralistische Systemarchitektur vorschreibt, ist die Verteilung der Daten darüber hinaus auf mehreren Physical Devices zulässig (Abbildung 3.7). In einem solchen Fall ist die Implementation eines oder mehrerer Clients neben dem Server im selben

Physical Device denkbar, so dass ein Datenaustausch innerhalb dieses Netzwerks möglich wäre. Schlussendlich handelt es sich also immer um einen Zugriff eines Clients auf einen Server, das auf einer Rechnerhardware (Physical Device) ausgeführt wird.

3.5.2 Servicemodelle

Um den Datenaustausch nach [IEC400-25-1] und [IEC400-25-3] von und zum Informationsmodell zu realisieren, werden Servicemodelle zur Verfügung gestellt, die die notwendigen Services zur Verfügung gestellt. Diese Services (z.B. Get, Set, Control, Query, Report) werden seitens des Servers angeboten und können von einem oder mehreren Clients für den Zugriff auf die instanzierten Informationen genutzt werden (Abbildung 3.8). Es handelt nach [IEC400-25-1] um die Services für:

- die Kontrolle über externe Operational Devices oder interner Funktionen des Devices,
- die Beobachtung des Prozesses und seiner Daten und
- das Management der Devices sowie den Zugriff auf das Datenmodell der Windenergieanlage.

Darüber hinaus werden Funktionen zum Reporten und Loggen bereit gestellt, die durch serverinterne Ereignisse gesteuert werden. Dabei werden Informationen direkt an einen Client versandt oder für eine spätere Verwendung gespeichert.

Grundsätzlich wird der Umfang der Services in die zwei Gruppen der

- Operational Functions und
- Management Functions

unterschieden. Der erstgenannten Gruppe werden Servicemodelle zuordnet, die Aufgaben zur Authorisation, Steuerung, Beobachtung und Reporten bzw. Loggen realisieren. Die zweit genannte Gruppe umfasst Servicemodelle zur Diagnose, Benutzer- und Zugriffverwaltung, Einstellung sowie zur Zeitsynchronisation.

Die Zuordnung findet sich in [IEC400-25-3] Table 1 den von aussen nutzbaren ACSI Servicemodellen gegenübergestellt. Die ACSI-Schnittstelle wird nachfolgend vorgestellt.

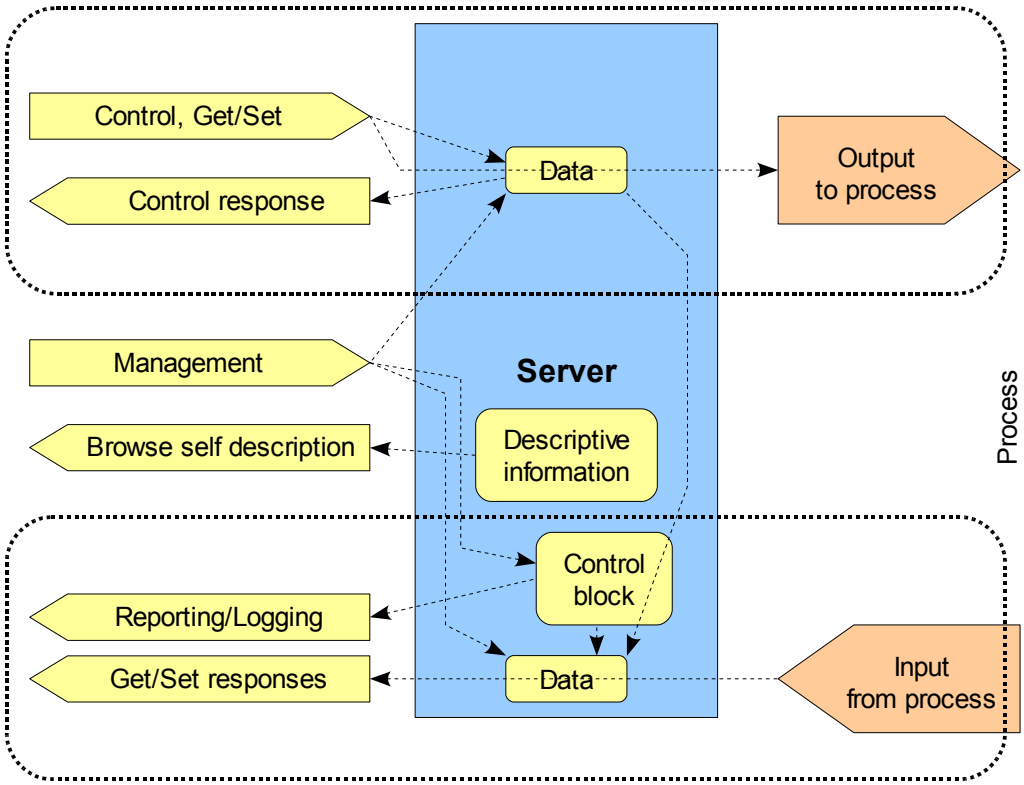

Abbildung 3.8: Servicemodelle des Datenaustauschmodells nach [IEC400-25-1]

3.5.3 Das Abstract Communication Service Interface

Zwecks Kommunikation zwischen den realen Komponenten und den Kommunikationspartnern außerhalb wurde das Abstract Communication Service Interface (ACSI) definiert. Basisdefinitionen finden sich in [IEC850-7-1] und [IEC850-7-2]. Wie in Abbildung 3.9 dargestellt, setzt sich das ACSI setzt sich aus mehreren Komponenten zusammen.

Das Physical Device mit einer Kommunikationsschnittstelle wird durch den Server gebildet. Er beinhaltet mindestens ein Logical Device mit je mindestens einem Logical Node, das sich von den beiden Logical Nodes LPHD und LLN0 zur Beschreibung des Physical Devices und des Logical Devices unterscheidet. Logical Nodes enthalten Daten, die individuell oder in Datengruppen gelesen werden könne. In letztgenanntem Fall ist die Auswahl einiger Daten aus einem oder mehreren Logical Nodes erlaubt, um sie in einer benannten Gruppe zusammenzufassen und mit einer einzigen Anweisung zu bearbeiten.

Darüber hinaus sind die ist die Aussendung von Reports, die Bearbeitung von gespeicherten Logs sowie die Reaktion auf Befehle implementiert. Üblicherweise sind Befehle, Get/Set und Log-Kommunikation (Abfragen/Antworten) bidirektional, während Reporte nur unidirektional übermittelt werden. Der enthaltene Log Control Block und Report Control Block dienen dem

Spezifizieren des Dateninhalts sowie der Bedingungen für ein Logging bzw. Reporting. Dabei darf von entsprechenden Fähigkeiten des Physical Devices, sofern sie über solche verfügen, Gebrauch gemacht werden.

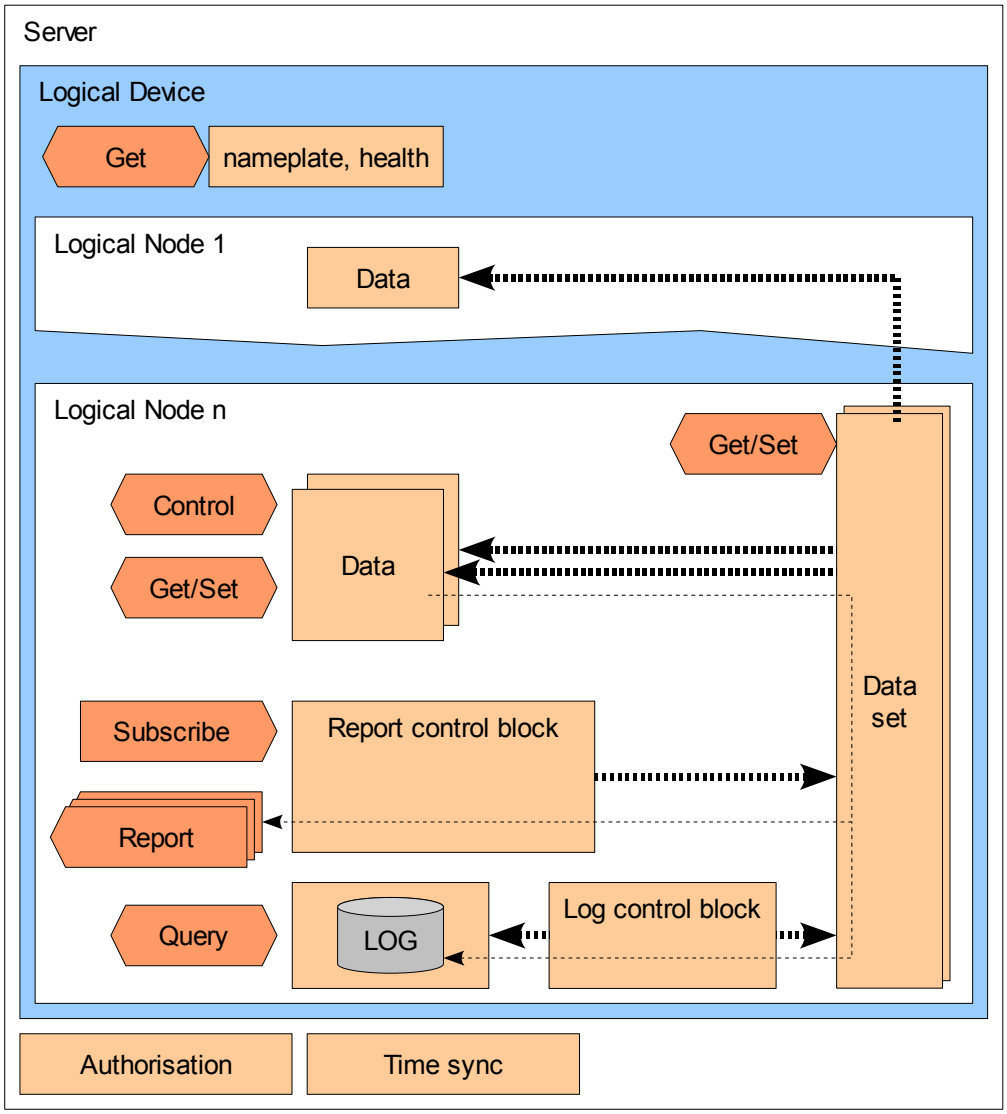

Abbildung 3.9: konzeptionelles Datenaustauschmodell nach [IEC400-25-1]

Neben den über die ACSI Schnittstelle verfügbaren Services, die unter Berücksichtigung von Echtzeitaspekten genutzt werden können (z.B. get, set) sind auch solche ohne diese Notwendigkeit verfügbar (Logging). Im Fall der Reportservices sind existiert die Unterscheidung in Unbuffered Reporting und Buffered Reporting. Die erstgenannte Option kann unter Echtzeitanwendungen genutzt werden.

3.5.4 Modellierungsregeln der ACSI-Services

Alle ACSI-Services unterliegen Modellierungsregeln, die demAnwender bekannt sein müssen, um

sie nutzen zu können. Die Services sind generell definiert durch:

- einen Satz Regeln für die Definition der Nachrichten (Messages),
- den Abfragenparametern sowie der zu erwartenden Rückgabewerte oder Fehler und
- vereinbarte Aktionen, die der Service auch ohne Ereignis des Prozesses ausführt.

Im Wesentlichen sind alle Nachrichten entweder Abfrage, positive oder negative Antwort. Positive Antworten werden zurückgeliefert, wenn eine Aktion ohne Fehler ausgeführt wurde oder wird. Tritt während einer Aktion ein Fehler auf oder kann die Aktion nicht ausgeführt werden, fällt die Antwort negativ aus und liefert neben den Rückgabewerten auch Fehlerwerte zurück. Jede der beiden möglichen Antworten hat also eine gewisse Zahl an Parametern und ggf. Fehlern, wie in Abbildung 3.10 dargestellt.

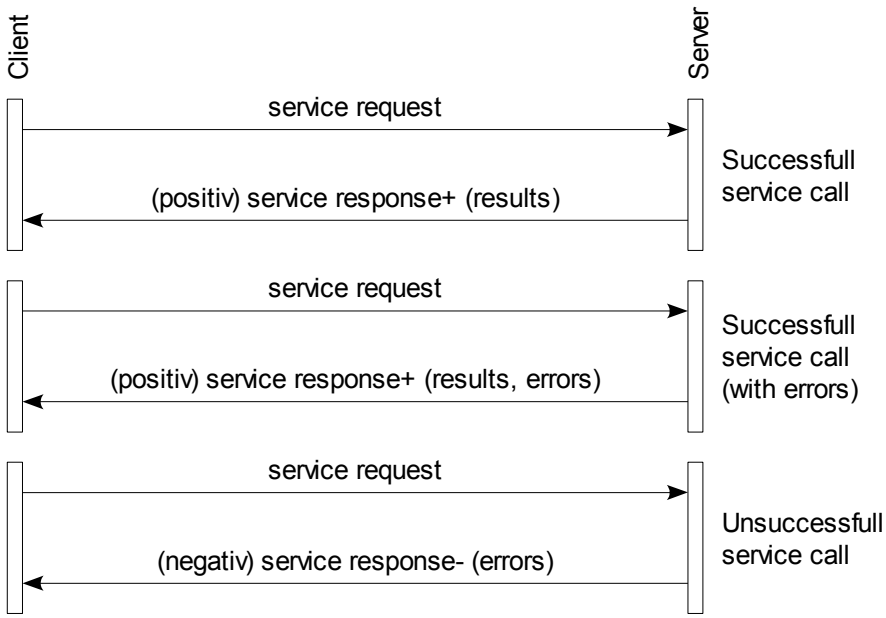

Abbildung 3.10: Sequenzdiagramm für Serviceaufrufe nach [IEC400-25-1]

Alle Services werden mit mindestens einer Parametertabelle in der Art wie Tabelle 3.6 (abstrakte Darstellung) dokumentiert. Enthalten sind die Parameter, die jeder Anfrage zuzuordnen sind sowie die Rückgabewerte bei erfolgreicher oder nicht erfolgreicher Ausführung des Services.

Die Nachrichten arbeiten mit den Attributen der Datenobjekte. Dies können Klassen des Datenmodells sein (z.B. Logical Node oder Datenattribute) oder Control Blocks, beispielsweise für die Reporting bzw. Logging Aktivitäten.

Parameter name
Request
Parameter 1...
Parameter n
Request+
Parameter 1...
Parameter n
Request-
Parameter 1...
Parameter n

Tabelle 3.6: Service Tabelle nach [IEC400-25-1]

3.6 Zusammenhang zwischen Daten- und Datenaustauschmodell

Die in die vorhergehenden Abschnitten wurden das Daten- sowie das Datenaustauschmodell nach [IEC400-25-2] und [IEC400-25-3] betrachtet. Mit Blick auf den gewünschten Gesamtzusammenhang ist dies jedoch nicht ausreichend. Dies zu erreichen, sind zwei weitere Schritte notwendig.

Im ersten Schritt wird bei Betrachtung der Abbildung 3.6 deutlich, dass der Zusammenhang zwischen Physical Device, Logical Device und den hierarchisch unter lagerten Logical Nodes nur per pauschaler Definition festgelegt wurde. Diese in der IEC 61400-25 nicht erwähnten Glieder müssen aus der [IEC850-7-2] bezogen werden. Im Einzelnen bezieht sich dies auf die SERVER Class ([IEC850-7-2], S 24) und die LOGICAL-DEVICE Class ([IEC850-7-2], S 34). Diese beiden Klassen enthalten Attribute, die den Zusammenhang zu der untergeordneten Hierarchie (Logical Device [0...n] bzw. Logical Node [3...n]) sicherstellen.

Darüber hinaus können noch weitere Klassen zur Anwendung kommen, die anstatt einer ausführlichen Erklärung nur als Referenz in der IEC 61400-25 Erwähnung finden, jedoch für die späterer Implementation erforderlich sind. Als Beispiel sei hier die TWO-PARTY-APPLICATION-ASSOCIATION Class genannt, die u.a. Daten und Services (z.B. Associate, Abort und Release; siehe [IEC400-25-3], Table 3 oder [IEC850-7-2], Table 12) für die allem voran stehende Authentifizierung enthält.

Nach der Herstellung des Zusammenhanges des Datenmodells, ist ein zweiter Schritt zur Anwendung der ACSI-Services zu tun. Die Informationen aus dem Datenmodell und die zugehörigen Services können in der jeweiligen Klasse integriert werden. So sind der Service

GetServerDirectory in der SERVER-Class zu finden und GetLogicalNodeDirectory ist Teil eines Logical Nodes (Wxxx Class).

Bezüglich des Datenmodells bauen alle Klassen aufeinander auf. Die Daten der jeweils untergeordneten Ebene sind für den Aufbau der jeweiligen Klasse notwendig. Man spricht in diesem Fall von einer Komposition, wie im Klassendiagramm in Abbildung 3.11 dargestellt. Im Gegensatz zu einer Aggregation existiert die übergeordnete Klasse nicht, wenn nicht mindestens eine untergeordnete Klasse vorhanden ist.

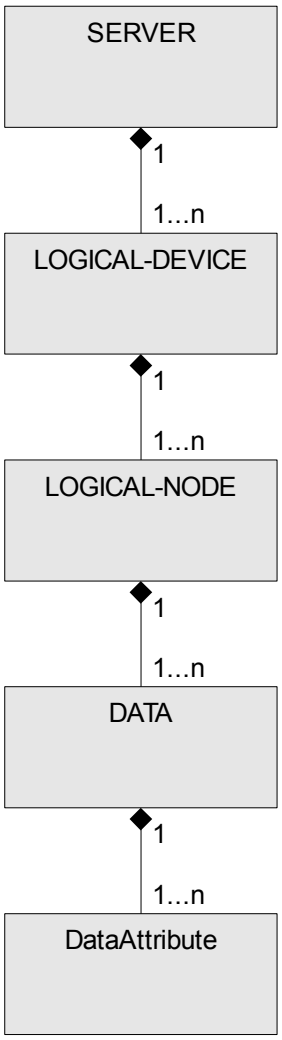

Abbildung 3.11: Klassenmodell des Datenmodells nach [IEC850-7-2]

3.7 Kommunikationsprofile nach IEC 61400-25-4

Der IEC 61400-25 Standard sieht als dritten wichtigen Teil das Mapping zu Kommunikationsprofilen vor. Dies geschieht über das Specific Communication Service Mapping (SCSM). Es definiert die Weise, auf die die Services und Modelle, wie Server oder Logical Device, in einem spezifischen Kommunikationsprofil abgebildet sind. Dabei soll das SCSM unabhängig vom Kommunikationsstack und der enthaltenen Applikationsprotokolle sein (Abbildung 3.12). Die Applikationslayer bewerkstelligen die konkrete Netzwerkadaption. In Abhängigkeit von der Art des Mappings, weisen sie eine unterschiedlich hohe Komplexität auf. Auch sind nicht alle ACSI-Services durch alle Mappings in kompletten Funktionsumfang realisierbar. In [IEC400-25-4] werden die ACSI-Services denen der fünf vorgesehenen Mappings gegenüber gestellt. Im Vergleich der Umsetzung der Mappings untereinander, sollten gleiche Services, die abzubilden sind, ein nach außen gleiches Verhalten vorweisen. In [IEC400-25-4], Table 1 findet sich eine Auflistung, welcher ACSI-Service im jeweiligen Mapping vorgesehen ist.

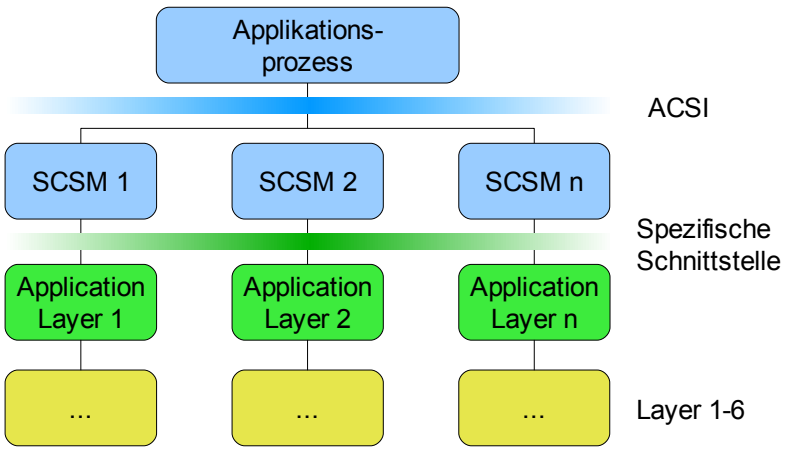

Abbildung 3.12: ACSI Abbildung auf verschieden Kommunikationsprofile nach [IEC400-25-1]

Die Architektur der Mappings kann verschieden weit in das OSI-Schichtenmodell greifen (Abbildung 3.13). Das OSI-Referenzmodell stellt das gängige Modell für Kommunikations-anwendungen dar. Dabei werden sieben Schichten definiert, denen alle Aufgaben einer Kommunikation von der Applikation (Layer 7) bis zur physikalischen Ebene (Layer 1) zugewiesen werden. Die IEC 61400-25 orientiert sind dabei am TCP (Transmsission Control Protocol, implementiert den OSI-Layer 4) bzw. dem IP (Internet Protokoll, implementiert den Layer 3) als niedrigste zu verwendende Protokolle. Die Layer 1 und 2 des OSI-Schichtenmodells werden in dieser IEC nicht betrachtet.

In der [IEC400-25-4] werden die notwendigen Nachrichten für die fünf vorgesehen Protokollstacks dokumentiert, um die notwendigen zum Informationsaustausch zwischen Client und Server zu realisieren. Die durch die IEC 61400-25 unterstützten Mappings sind:

- Mapping zu SOAP basierten Web Services

  Das Simple Object Access Protokoll dient zum Nachrichtenaustausch auf Basis von XML (Extensible Markup Language) in verschiedenartigen Anwendungen, z.B. im Rahmen von Web Services. Web Services bedienten Clients mit angefragten Informationen innerhalb von Softwaresystemen.

- Mapping zu OPC/XML-DA

  OPC (OLE for Process Control) ist ein verbreiteter Standard für Automatisierungssysteme über Herstellergrenzen hinweg. Er existiert in mehreren Varianten und Entwicklungsstufen. Aus dem OPC DA (Data Access) für die Übermittlung in Echtzeit hat sich OPC/XML DA fortentwickelt.

- Mapping zu IEC 61850-8-1 MMS

  Die Manufactoring Message Specification ist eine vorwiegend für die flexible und objektorientierte Integration von Automationssystemen im Produktionsbereich gedachtes Protokoll.

- Mapping zu IEC60870-5-104

  Hierbei handelt es sich um ein allgemeines Protokoll zur Kommunikation zwischen Leittechnik und Substations, das Funktionen z.B. zur Anwendung mit SCADA-Anwendungen anbietet.

- Mapping zu DNP3

  Das allgemeine Protokoll zur Kommunikation zwischen Leittechnik und Substations wird vorwiegend in der Fernwirktechnik eingesetzt.

Da der erwähnte Inhalt dieses IEC-Teils nicht mehr im Fokus dieser Arbeit liegt, wird auf ausführliche Erklärungen zugunsten dieser kurzen Beschreibung der wesentlichen Eigenschaften verzichtet.

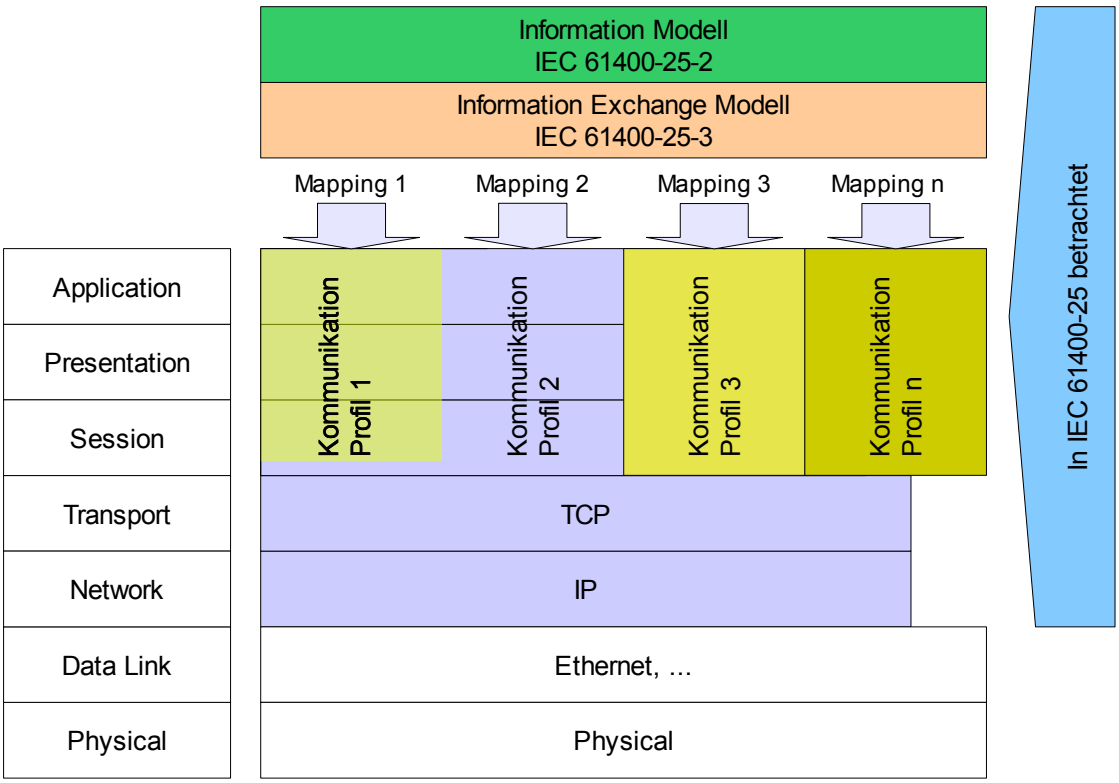

Abbildung 3.13: Kommunikationsprofile nach [IEC400-25-4]

4 Entwurf und Teilimplementierung der IEC 61400-25

4.1 Entwicklungsprozess

4.1.1 Entwicklungsschritte

4.1.1.1 Allgemeines zum Entwicklungsprozess

Jeder Entwicklungsprozess zerfällt in einzelne Entwicklungsschritte. Sie umfassen alle Maßnahmen von der Idee bis zur Umsetzung des Produktes. Es sei hier vor angekündigt, dass in diesem Fall das Produkt in Form einer Software angestrebt wird. Die Entwicklungsschritte werden hier nach ihrer Funktion und später ausführlicher und auf diese Arbeit bezogen beschrieben. Es wird an dieser Stelle kein Anspruch auf Vollständigkeit erhoben, da die Schritte in Abhängigkeit des Projektumfanges auch erweitert und angepasst werden können. Beispielsweise wird in [SEGU06] eine Projektentwicklung vorgestellt, in der eine Unterteilung in ein statisches und dynamisches Modell (Objektstruktur/Ereignisstruktur) vorgenommen wird.

4.1.1.2 Nutzenanalyse

Die Nutzenanalyse hat die grundlegende Aufgabe zu klären, warum ein Entwicklungsprozess angestoßen werden soll. Die Gründe einer Befürwortung sind typischerweise in der Steigerung der Effizienz von bestehenden Systemen oder Resultat einer Kostenrechnung. Weitere Gründe können die Bewältigung neuer Herausforderungen, die Erfüllung von Kundenwünschen oder Prozesserfordernisse sein. Weniger wahrscheinlich, aber doch möglich, ist auch ein Abbruch des Entwicklungsprozesses.

Neben der Entscheidung für oder gegen den Beginn einer Entwicklung beinhaltet die Nutzenanalyse zumeist auch die grobe Planung des weiteren Entwicklungsfortgangs, z.B. über die Wahl der vorgenannten Vorgehensmodelle und des angestrebten Lösungsweges.

4.1.1.3 Anforderungsanalyse

Ist die Entscheidung für den Beginn eines Entwicklungsprozesses gefallen, besteht der Inhalt dieses Schrittes in der vollständigen Spezifizierung der Ziele, oft in Form je eines Lasten- und Pflichtenheftes. Diese Maßnahme bildet die Grundlage für alle folgenden Schritte und beinhaltet alle Sichten der beteiligten Parteien. Deshalb ist eine Zusammenarbeit aller Prozessbeteiligen zwingend notwendig, um Projektverzögerungen und Rückschritte innerhalb des Vorgehensmodells durch Fehler und Unzulänglichkeiten zu vermeiden.

Die formulierten Ziele beinhalten den Funktionsumfang und ggf. Vorentscheidungen zum Realisierungsweg bzw. Auswahl der Lösungsmittel (z.B. vorbestimmte Software), die sonst in den nachfolgenden Entwicklungsphasen festgelegt würden. Es kommen geeignete Werkzeuge zur Anwendung, die komplexe Vorgänge geschäftlicher oder funktioneller Art, beherrschbar machen. In dieser Arbeit wird den Darstellungsmöglichkeiten der von UML (siehe [SEGU06]) Gebrauch gemacht. Neben der funktionellen Definition der Entwicklungsziele sind weiterhin Angaben zum Projektablauf, wie Rollenzuweisungen und Verantwortlichkeiten, aufzunehmen.

4.1.1.4 Entwurf

Die Entwurfsphase gehört den konkreten Überlegungen zur Umsetzung der Anforderungsanalyse. Spätestens jetzt werden wird eine geeignete Systemarchitektur modelliert, die geeignet ist, allen gewünschten Anforderungen gerecht zu werden. Unter einer Systemarchitektur versteht man die Identifikation der Objekte, des dynamischen Verhaltens sowie ihrer Beziehungen zueinander. Gilt es neben einer Software, wie in diesem Fall, auch eine Hardware zu realisieren, wird zusätzlich in Softwarearchitektur und Hardwarearchitektur unterschieden. Zum Entwurf gehören aber auch die Beachtung des Systemumfeldes. Dazu sind die Zielplattform, Schnittstellen und Interaktionen mit dem Umfeld zu bestimmen. Es ist das Ziel zu klären, was das zu erstellende System leisten soll.

Im Idealfall kann das Ergebnis der Entwurfsphase in der Implementierungsphase direkt und ohne Anpassungen umgesetzt werden.

4.1.1.5 Implementierung

Aufgabe der Implementierung ist es, die Ergebnisse des vorgenannten Entwurfsschrittes produktiv umzusetzen. Dazu gehört die vorherige Wahl der Werkzeuge, wie z.B. Programmiersprache und der Entwicklungstools.

4.1.1.6 Test/Validierung

Ein betriebsbereites System liegt jetzt vor. Mit Hilfe simulierter Schnittstellen und Daten wird das System in Testszenarien auf seine vorbestimmte Funktion getestet und ggf. korrigiert.

4.1.1.7 Betrieb

Das System nimmt nach einer Inbetriebnahmezeit seinen normalen Betrieb auf. Nachträglich gewünschte Änderungen oder Ergänzungen können zum Anstoß eines neuen Entwicklungsprozesses führen. In den normalen mit inbegriffen sind jedoch Wartungsmaßnahmen.

4.1.2 Vorgehensmodelle

Die Abfolge der Entwicklungsschritte erfolgt nach einem vorbestimmten Vorgehensmodell. Derer unterscheidet man oft nach Merkmalen, wie z.B. ein Entwicklungsschritt zum nächsten übergeht, ob die Schritte rekursiv zu durchlaufen sind oder ob sie überhaupt als solche zu finden sind. Hier werden mögliche Vorgehensmodelle vorgestellt.

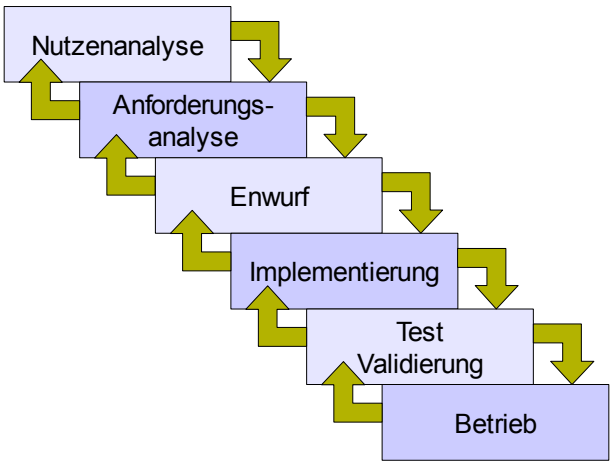

Abbildung 4.1: Wasserfallmodell nach [BRPK08]

Beispielsweise sieht das relativ robuste und einfache Wasserfallmodell (Abbildung 4.1) vor, alle Ergebnisse eines Schrittes in den jeweils nachfolgenden Schritt übergehen zu lassen. Die Vorgehensweise berücksichtigt dabei allerdings nur Rücksprünge zum jeweils letzten Schritt. Dies kann zum Nachteil werden, wenn man den abschließenden einzelnen Test nahe dem Ende der Entwicklungskette betrachtet und die Ursachen für Unzulänglichkeiten weiter vorn zu suchen sind. Eine typische Eigenschaft des Wasserfallmodells ist zudem, dass alle Entwicklungsschritte (evtl. mit anderem Namen) bekannt und als solche eindeutig zu identifizieren sind. Es entfällt jedoch die Benennung der Rollen und Methoden. Im Zuge steigender Ansprüche an die Qualität einer Software

wiegt der Mangel an Testmöglichkeiten allerdings schwer. Abschließend kann man sagen, dass einfachere Projekte mit Hilfe des Wasserfallmodell gut zu bewältigen sind.

Alternativ existieren Ablaufprinzipien, wie etwa das V-Modell (Name rührt von der Darstellung, siehe Abbildung 4.2). Es erweitert einerseits das Wasserfallmodell um Testmöglichkeiten in kleinen Einheiten (Modulen), zu denen jeweils eigene Ergebnisse definiert sind. Fehlentwicklungen und Abweichungen von den ursprünglichen Zielen können durch die Gewichtung von Tests in Modulen eher vermieden werden. Daher ist eine Eignung für größere Entwicklungen nahe liegend. Andererseits liegt die Orientierung des Modells zusätzlich mehr bei den organisatorischen Maßnahmen, also was, wie und womit zu tun ist. In einer weiteren Verfeinerung kann das V-Modell in die vier Submodelle Projektmanagement, Systemerstellung, Qualitätssicherung und Konfigurationsmanagement unterteilt werden.

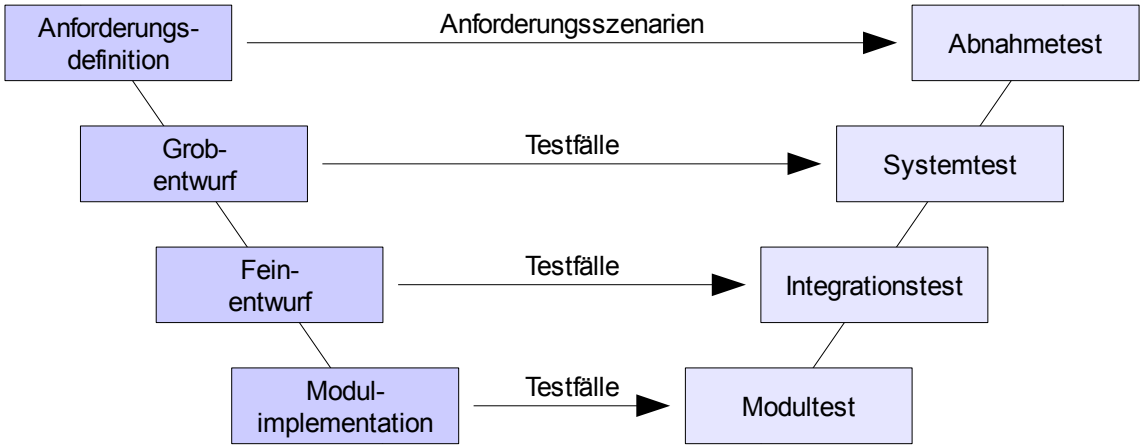

Abbildung 4.2: V-Modell nach [BRPK08]

Neben den erwähnten Vorgehensmodellen existieren noch solche, die zu Beginn der Entwicklung noch kein definiertes oder nur ein grob skizziertes Ziel aufweisen. In solchen Fällen muß die Definition zwangsweise während der Entwicklung vorgenommen werden.

Als dritter Weg soll deshalb das Spiralmodell erwähnt werden. Es erlaubt ausdrücklich eine rekursive Arbeitsweise derart, dass die Entwicklungsschritte in mehreren Zyklen durchlaufen werden. Dabei werden die aktuellen Ergebnisse zur Verfeinerung der, anfänglich grob formulierten, Projektziele genutzt. Es ergibt sich zudem die Möglichkeit, Teilergebnisse zwischenzeitlich auf ihre Brauchbarkeit zu testen und ggf. Alternativen zur Anwendung kommen zu lassen. Zum Ende eines Zyklusses wird jeweils nächste geplant. Dieser Vorgang wird wiederholt bis das gewünschte Ergebnis zur Verfügung steht. Dieses Modell eignet sich daher, wie schon erwähnt, vornehmlich zur Bewältigung vorher nicht präzise bestimmbarer Projektziele. Verglichen mit dem Wasserfallmodell

bekommen qualitätssicherende Maßnahmen mehr Gewicht und tragen zu mehr Produktqualität bei. Dies beinhaltet auch zusätzliche Maßnahmen der Risikoanalyse.

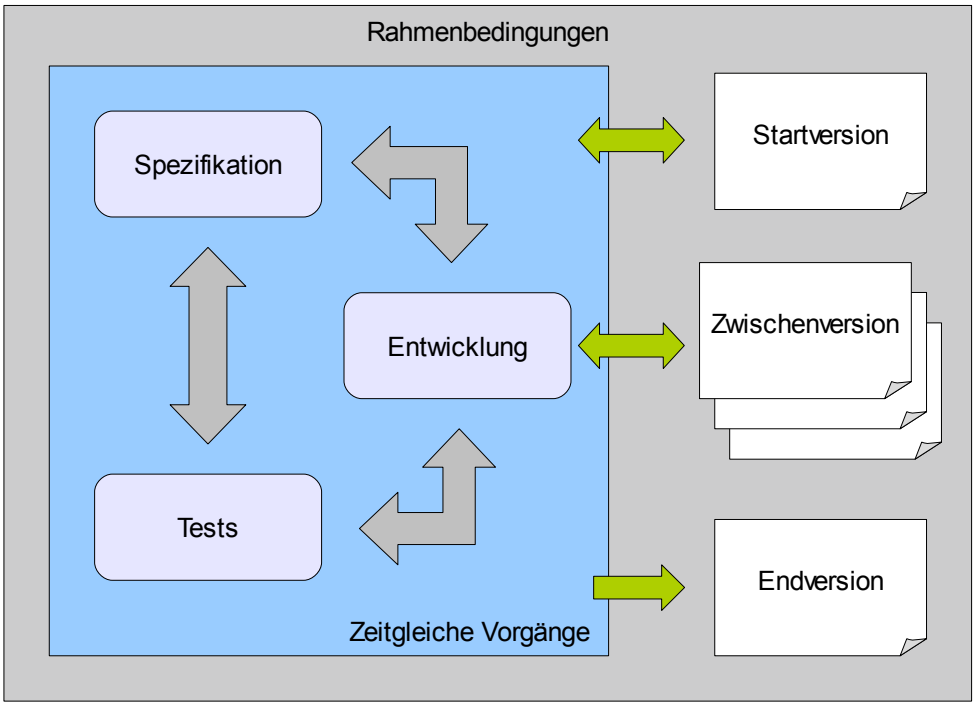

Abbildung 4.3: Evolutionäre Softwarentwicklung [BEHA04]

Als letztes Vorgehensmodell werfen wir einen Blick auf das evolutionäre Modell (Abbildung 4.3). Es berücksichtigt auf gleiche Weise nicht oder grob vorhandene Zieldefinitionen. Auch hier werden verschieden Zwischenstationen genutzt, um die finale Zieldefinition zu erfassen. Dabei wird von einer Startimplementierung Gebrauch gemacht, die nach einigen Zwischen- in eine endgültige Version überführt wird. Bezeichnend ist, daß die bekannten Entwicklungsstufen nicht als solche zu identifizieren sind und die zugehörigen Maßnahmen zeitgleich statt finden. Aufgrund dieser Komplexität eignete sich dieses Vorgehensmodell nur für kleine Projekte und erfahrene Teams.

Damit soll der kurze Blick auf vier wesentliche Vorgehensmodelle zur Softwareentwicklung abgeschlossen werden. Es existieren weitere, z.T. sehr spezifische, Modelle, die hier jedoch keine Anwendung finden.

4.2 Anforderungen

4.2.1 Motivation

Wie bereits in den vorhergehenden Abschnitten dargelegt ist es Ziel der Anforderungsanalyse, die Projektziele zu ermitteln. Im bevorzugten Fall kann eine genaue Definition dessen durch Erstellung je eines Lasten- und eines Pflichtenheftes erfolgen. Sind diese Bedingungen nicht erfüllt, bietet sich die Anwendung des vorgenannten Vorgehensmodells der evolutionären Softwareentwicklung an. Die Ziele werden für die jeweils nächste Projektstufe in Angriff genommen und bauen dabei auf die jeweils vorhergehende Stufe auf.

Die Abteilung Competence Center AUTOMATIONWORX (CCAX) der Firma Phoenix Contact Electronics GmbH betreut Automationsprodukte aus dem eigenen Haus und anderer Hersteller. Es werden eine Reihe von Sparten bedient. Ein Schwerpunkt bildet die Windbranche. Zum Lieferumfang der Fa. Phoenix für Anlagenhersteller gehören u.a. Installationsmaterial und Steuerungen. Letzteres wird als Zielplattform für Softwarelösungen im Zusammenhang mit dieser Arbeit besonderes Augenmerk finden. Kunden, die dieses Produkt zur Anlagensteuerung verwenden, äußern in zunehmendem Maße Interesse an einer Nutzung unter Berücksichtigung der IEC 61400-25, um ihrerseits dem Markttrend zu folgen. Recherchen im Marktumfeld haben ergeben, dass erste Monitoringprogramme von unabhängigen Softwarefirmen angeboten werden. In zunehmender Zahl bieten auch Mitbewerber eine Serverlösung mit ihren Steuerungen an. Jedoch ist kein Fall bekannt, in dem eine Praxistauglichkeit bereits unter Beweise gestellt wurde.

Aus diesen Gründen erklärt sich Motivation der Fa. Phoenix, dem Markttrend mit einer Lösung zu begegnen, die für die Belange der Kunden geeignet ist. Da jedoch auf Kundenseite wie auch bei Fa. Phoenix nur geringe Kenntnis über die IEC 61400-25 und deren Anwendung vorhanden ist, kann eine detaillierte Definition der Projektziele nicht vorgenommen werden.

4.2.2 Anforderungen an die Lösung

Aufgrund der vorgenannten Gründe ist es also nicht möglich, eine präzise Zieldefinition abzugeben. Jedoch ist es unerlässlich, IEC 61400-25 auf ihre Umsetzbarkeit zu untersuchen, bevor eine finale Lösung definiert werden kann. Gemäß dem Modell der evolutionären Vorgehensweise wurden in einem ersten Schritt Ziele festgelegt, die es in einer ersten Stufe zu erreichen gilt. Diese Ziele sind:

1. Die Automatisierungsprodukte der Fa.Phoenix werden für eine Lösung verwendet. Damit ist die Steuerung RFC 470 PN 3TX und die dazugehörige Entwicklungsumgebung PCWORX gemeint. Der RFC 470 PN 3TX ist speziell mit Hinblick auf den Einsatz im Windbereich entwickelt worden.
2. Die aufgefundene Lösung soll minimal und einfach sein, so dass eine Basis für die nächste Entwicklungsstufe geschaffen wird.
3. Für andere Kollegen soll es möglich sein, Kundenlösungen bevorzugt ohne aufwendige Referenz des umfangreichen IEC 61400-25 Standards zu erstellen. Da in der zuständigen Abteilung CCAX Erfahrungen in Structured Text (Programmiersprache ST ist Teil der IEC 61131) vorhanden sind, sollen alle Programmieraufgaben allein mit Kenntnis von ST zu lösen sein.

Die formulierten Ziele sind sehr weit gefasst, reichen jedoch für die angedachte Formulierung der Ziele der ersten Entwicklungsstufe aus. Im nächsten Schritt werden die Voraussetzungen und Startbedingungen auf ihre Vereinbarkeit hin untersucht.

4.3 Randbedingungen

4.3.1 Voraussetzungen

4.3.1.1 Der RFC 470 PN 3TX

Wie bereits erwähnt, ist der RFC 470 PN 3TX (nachfolgend nur RFC genannt) die derzeit bevorzugte Steuerung für Windenergieanlagen bei Phoenix Kunden. Diese Kompaktsteuerung ist für den Schaltschrankeinbau bestimmt und besitzt eine Aufnahme für die Hutschienenmontage.

Zur Kommunikation mit der Umwelt besitzt die Steuerung eine Reihe von Schnittstelle. Der Zugang zur E/A-Ebene steht über eine Interbusschnittstelle zur Verfügung, so dass der RFC als Interbusmaster agieren kann und an die Leitebene angeschlossen wird. Zusätzlich besitzt der RFC eine Ethernetschnittstelle. Über diese werden TCP/IP und UDP/IP unterstützt. Ebenso ist ein Anschluss an Profinet IO möglich. Die Konfiguration ermöglicht dabei den wahlweisen Modus als Profinet IO-Device oder Profinet IO-Controller. Auch über Ethernetschnittstelle kann die Anbindung an die Leitebene erfolgen. Weitere Details finden sich in [RFC09].

Das Interbussystem war in der Vergangenheit in Windenergieanlagen der Phoenix-Kunden

dominant. In zunehmendem Maße werden jedoch auch Ethernet basierte Bussysteme wie Profinet eingesetzt.

Abbildung 4.4: RFC 470 PN 3TX (Phoenix Contact Electronic GmbH)

4.3.1.2 Kurzbeschreibung Interbus und Profinet

Beim Interbus handelt es sich um Sensor/Aktorsystem mit der erweiterten Möglichkeit, komplexe Technologiesteuerungen einzubinden. Topologisch wird dieses Bussystem als Ringsystem ausgeführt. Physikalisch wird er Bus auf Basis der RS485 Schnittstelle mit einer maximalen Datenübertragungsrate von 500kByte/s betrieben. Dabei beträgt der größte Punkt zu Punkt Abstand 400 Meter. Es gibt die Möglichkeit untergeordnete Peripheriebussysteme, mit einer maximalen räumlichen Ausdehnung von bis zu 10 Metern und nicht mehr als 8 Teilnehmern oder einen Installationsfernbus mit bis zu 50 Metern in den Fernbus einzubinden. Der Fernbus kann eine Ausdehnung von bis zu 13 km erreichen, dies aber unter Verwendung Busklemmen, die auch als Repeater fungieren. Der Einsatz von Busklemmen erweist sich auch als Vorteil, wenn Fehlerstellen durch Segmentierung isoliert werden sollen oder auf eine ein anderes Trägermedium umgesetzt wird. Insgesamt ist die Anzahl der Busteilnehmer auf 256 beschränkt. Die Verdrahtung des

Interbussystems erfolgt als geschirmter Fünfdraht für den Fernbus und ungeschirmter Zweidraht (Signale und Energie) für den Peripheriebus.

Im Gegensatz zum Interbus ist das Profinet kein Bussystem an sich. Es ist vielmehr als umfassendes und komponentenbasiertes Konzept zu bezeichnen. Dabei ist der Profibus nur ein Bestandteil unter anderen. Die Entwicklung wurde durch die Profibus Nutzerorganisation (PNO) u.a. mit den Zielen angestoßen, Profibusnetze zu integrieren und Ethernet als bevorzugte Basis für eine Vernetzung von der Feldebene zur Unternehmensleitebene zu nutzen.

Das Konzept unterteilt sich in:

- Engineering, für die einheitliche Entwicklung der Profinet-Objekte über eine einheitliche Schnittstelle;
- Kommunikation, für die Herstellung der Kommunikation zwischen den Profinet-Objekten mittels des Active Control Connection Object (ACCO) und DCOM als Protokoll;
- Einbindung von Profibus uns anderen Bussystemen über Interfaces, Gateways usw. als Profibus-Objekte sowie
- I/O-Integration via Ethernet, für die Einbindung schneller und zyklischer Peripherie über ein, neben dem DCOM existierenden, Echtzeitprotokoll.

Um die beschriebenen Eigenschaften des RFC nutzen zu können, stellt Phoenix eine umfangreiche Entwicklungsumgebung PCWORX bereit.

4.3.1.3 IEC 61131

Die IEC 61131 steht seit 1993 für die weitgehend herstellerunabhängige Programmierung von speicherprogrammierbaren Steuerungen zur Verfügung. Für die Verwendung in unserem Fall ist der Teil 3 von besonderer Bedeutung, in der die fünf möglichen Programmiersprachen definiert werden. Zu diesen Programmiersprachen gehören der Hochsprachen ähnliche Strukturierte Text (ST), die Anweisungsliste (AWL), die Ablaufsprache (AS), der Kontaktplan (KOP) und die Funktionsbausteinsprache (FBS). Alle Sprachen sind gleichwertig eingeordnet und haben die Programmierung von Echtzeitsystemen zum Ziel.

Um diese Sprachen zum Einsatz bringen, sind einige Informationen zum IEC-Softwaremodell sinnvoll. Dieses Modell erlaubt die Einteilung in Programmorganisationseinheiten (POE) als kleinste Einheit. Damit sind Programme, Funktionsbausteine und Funktionen gemeint. Alle POE können Funktionsbausteine nach vorheriger Instanzierung und Funktionen direkt aufrufen.

Programme werden einer (Hardware) Ressource zugeordnet (Abbildung 4.5).

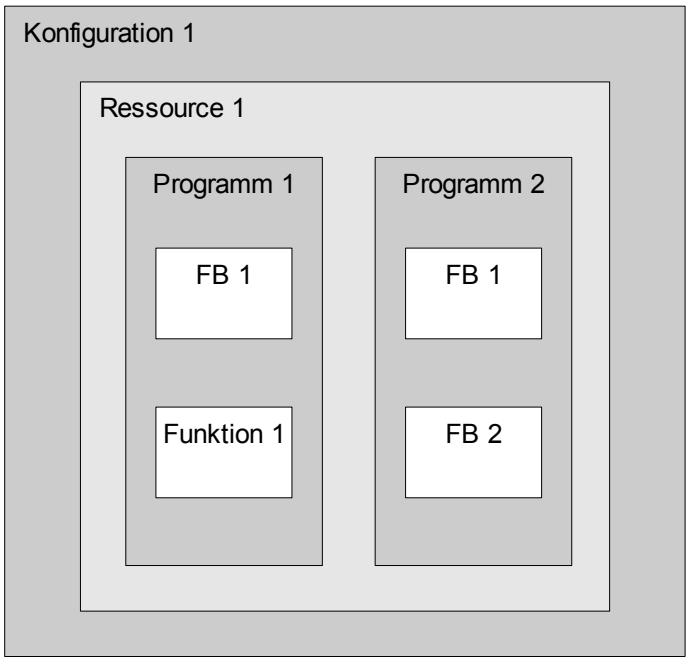

Abbildung 4.5: Softwaremodell der IEC 61131 nach [LOV07]

Diese Ressource in Form eines Prozessors steht jedoch nicht ständig zur Verfügung, sondern nur in gewissen zeitlichen Abständen. Man definiert sogenannte Tasks für die Aufgabenmanagment der Ressource. Auch Tasks sind Teil der IEC 61131-3. Es existieren Tasks von Typ Default, Cyclic, Event und System. Sind einer Task mehrere Programme zugeordnet, werden sie der Reihe nach ausgeführt. Tasks werden in zwei Gruppen, zyklische und ereignisgesteuerte Tasks, eingeteilt. Tasks vom Typ Default arbeiten mit einer geringen Priorität und starten nach Beendigung eines SPS-Zyklusses den nächsten. In Abbildung 4.6 entspräche ein SPS Zyklus der Zeitspanne von T1 bis T3. Da der Default-Task nur eine geringe Priorität hat, ist der SPS-Zyklus nicht von konstanter Länge, da er von einer anderen Task unterbrochen werden kann. Ist die unerwünscht, ergibt sich mit einer Task vom Typ Cyclic die Chance, Priorität und Zykluszeit in den Grenzen des technischen machbaren einzustellen. Damit ist eine deterministische Abarbeitung der Programme umsetzbar. Die zur Gruppe der ereignisgesteuerten Tasks vom Typ System und Event werden im Verbund mit SPS-internen (Kaltstart...) oder externen Ereignissen (Busfehler...) verwendet.

Die durch die Task gesteuerte Ausführung der Programme wiederholt sich also zyklisch bzw. ereignisgesteuert. Sind in dem Programm Funktionsbausteine instanziert oder werden Funktionen verwendet, so werden auch diese mit jedem Programmdurchlauf abgearbeitet. Programmteile, die nicht mit jedem Zyklus neu abgearbeitet werden sollen, müssen durch programmiertechnische Maßnahmen (Bedingungen usw.) davon ausgenommen werden.

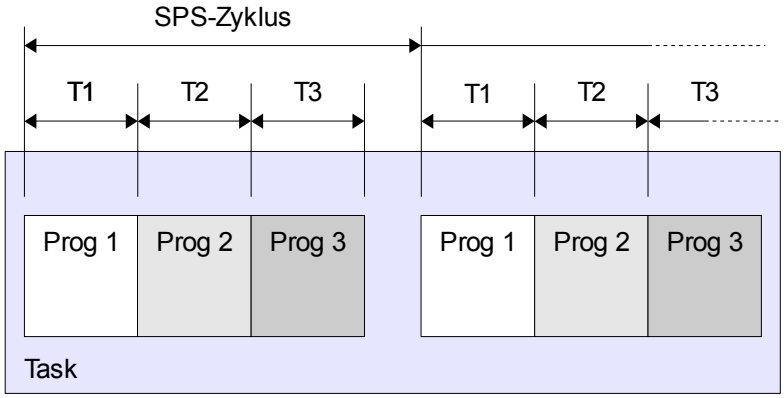

Abbildung 4.6: Mehrere Programme einer Task werden nacheinander abgearbeitet

Schlussendlich gilt es noch einen Blick darauf zu werfen, wie sich Daten nach der IEC 61131 kapseln lassen. Im Zusammenhang mit POE werden die Schnittstellenvariablen VAR, VAR_IN, VAR_OUT und VAR_IN_OUT betrachtet. Es existiert weiterhin die Unterscheidung in lokale und globale Variablen. Lokale Variablen sind nur innerhalb einer POE sichtbar. Im Projekt sichtbare globale Variablen können in Programmen und Funktionsbausteinen Verwendung finden, wo hingegen globale Variablen einer Programminstanz ausschließlich in Programmen nutzbar sind.

4.3.1.4 PCWORX

Als Entwicklungswerkzeug für die Steuerungsprodukte findet das hauseigene PCWORX Verwendung. Es liegt aktuell in der Version 5.2 vor und unterstützt die IEC 61131 mit seinen fünf Programmiersprachen nahezu vollständig. Die Verbindung zur Steuerung kann wahlweise über eine serielle Verbindung oder über Ethernet erfolgen.

Die PCWORX Oberfläche ist in Teilbereiche unterteilt. Der erste Teilbereich „Buskonfiguration“ befasst sich mit der Buskonfiguration für Profinet und/oder Interbus. Es ist möglich die vorhandene Busstruktur aufzubauen bzw. einzulesen und eine Editierung der Geräte vorzunehmen. Zu gehört auch die Diagnose der Automationshardware.

In einem zweiten Teilbereich „Prozessdatenzuordnung“ wird die Zuordnung der Prozessdaten vorgenommen. Dabei handelt es sich um die Verkettung der durch die Busteilnehmer zur Verfügung gestellten Daten mit (Prozessdatenobjekte) mit globalen Variablen zur Nutzung der Daten in der Steuerungsprogrammierung.

Der Bereich „IEC-Programmierung“ dient der Programmierung der Steuerung nach IEC 61131 sowie der Einstellung der Tasks. Es sind neben dem Default-Task auch die vom Typ Cyclic, Event

und System möglich. Zur Programmierung gehörte auch die Einbindung von fremder Codequellen, die durch Bibliotheken bereitgestellt werden. Darüber hinaus enthält es Werkzeuge zum Debuggen und für den Up-/Download zur SPS.

Der vierte Bereich „Projektvergleich“ dient dem Herausstellen der Unterschiede zweier offline gestellter Projekte.

Damit soll der kurze Blick auch die Entwicklungsumgebung abgeschlossen werden. Weitere Details werden fallweise in der jeweiligen Anwendung erläutert.

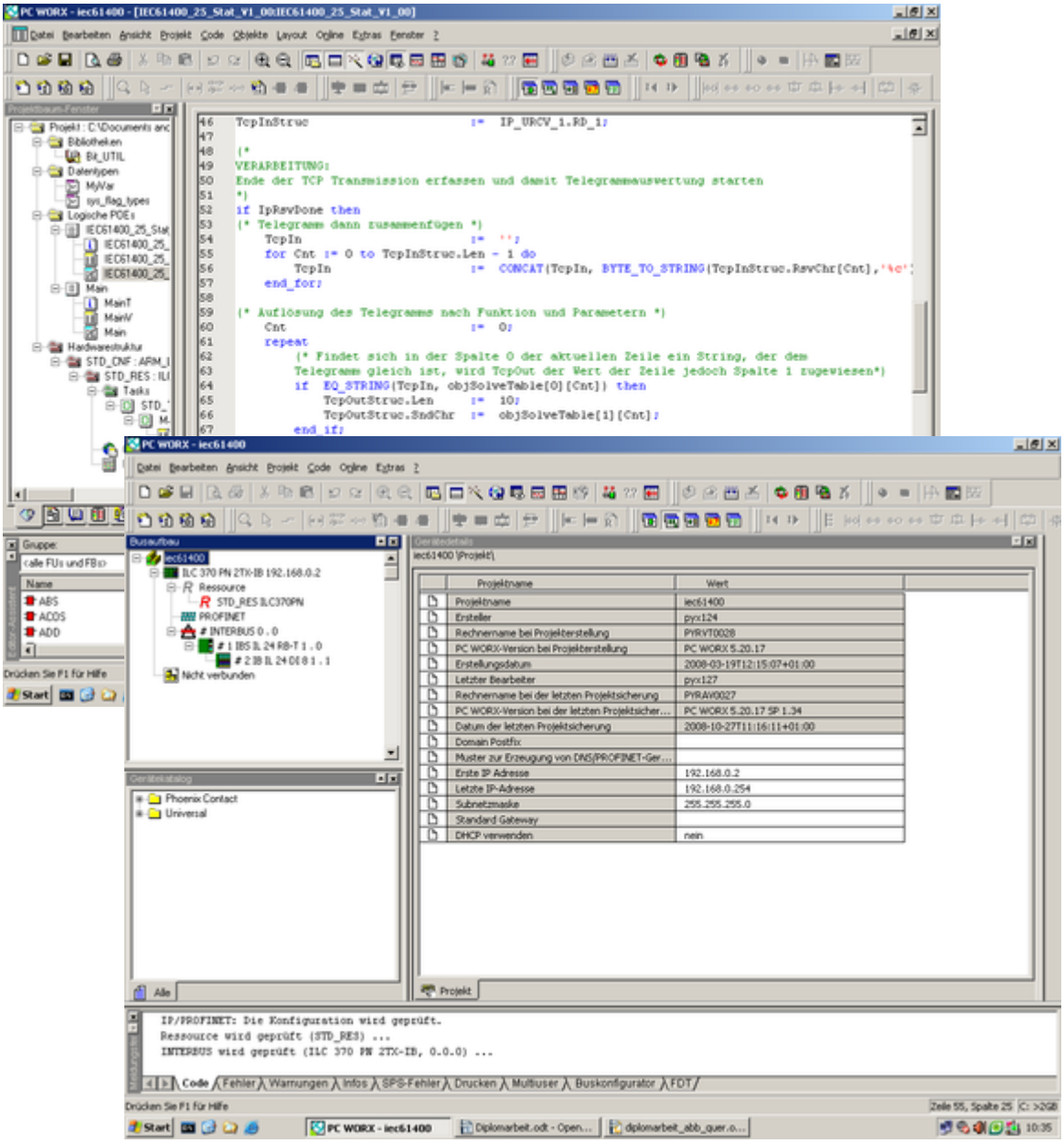

Abbildung 4.7: PCWORX Arbeitsbereiche für die IEC-Programmierung und die Buskonfiguration

4.3.2 Gegenübergestellung Objektorientierung vs. Strukturierter Text

4.3.2.1 Gründe der Gegenüberstellung

Um den Entwurf der IEC 61400-25 vornehmen zu können, ist die Beschreibung der objektorientierten Eigenschaften und Fähigkeiten, oder genauer, die des Daten- und Datenaustauschmodells notwendig. Dies ist in sofern essentiell, da alle diese Eigenschaften auch nach der Implementation im RFC auch als solche wiederzufinden und funktionell aktiv sein müssen. Der Vorgabe folgend, die Implementation bevorzugt in ST vorzunehmen, stellt sich die Frage, ob und wie die benannten objektorientierten Eigenschaften und Fähigkeiten überhaupt in einer prozeduralen Sprache wie ST umzusetzen sind. Um dieser Frage nachzugehen, werden die zur Verfügung stehenden Möglichkeiten des Strukturierten Textes nach IEC 61131zuerst untersucht.

4.3.2.2 Kurzbeschreibung wichtiger Merkmale von ST nach IEC 61131 (PCWORX)

Als hochsprachenähnliche Programiersprache weist ST die üblichen Möglichkeiten für die Umsetzung von z.B. Kontrollstrukturen auf. Von hauptsächlichem Interesse im Zusammenhang mit der Implementierung von Daten- und Servicemodellen ist die Frage, wie Felder, Strukturen und Funktionen eingesetzt werden.

Grundlage für jede Speicherung von Daten ist auch bei ST die Zuweisung eines Wertes zu einer Variable. Dazu ist es zuvor notwendig, die Variable mit einem Datentyp zu deklarieren. PCWORX stellt nahezu alle in der IEC 61131 definierten Datentypen zur Verfügung (siehe Tabelle 4.1, Tabelle 4.2 und Tabelle 4.3). Die Zuweisung trivialer Variablen ist in allen POE möglich.

Für den Aufbau komplexerer Zusammenhänge sind jedoch weitergehende Datenstrukturen nötig.

```
TYPE
        ARR_STR_0_100           :       ARRAY[0..100] OF STRING;
        ARR_STR_0_1             :       ARRAY[0..1] OF ARR_STR_0_100;
END_TYPE
```

Text 4.1: Definition zweier Arrays zu einer zweidimensionalen Feldstruktur

Eine erste Form der Strukturierung bildet ein eindimensionales Array. Die Möglichkeit, mehrdimensionale Arrays zu erstellen, ist in PCWORX an sich nicht zugelassen. Es ist jedoch ausweichend eine Verschachtelung zweier eindimensionaler Arrays machbar, wie Text 4.1 in dargestellt.

Ein Nachteil entsteht aus dem Sachverhalt, dass die Konstruktion mehrdimensionaler Arrays nur für

einen Datentyp zugelassen ist. Für die weitergehende Strukturierung der Daten zusammengesetzter Datentypen existiert daher die Möglichkeit, Variablen in einer Struktur zu bündeln. Dabei ist die Verwendung von unterschiedlichen Variablentypen genauso denkbar, wie die eines Arrays. Eine Steigerungsform ergibt sich aus der Verschachtelung mehrerer Strukturen ineinander. In Text 4.2 sind die Arrays ARR_B_0_100, ARR_B_0_1460 sowie die Struktur UDT_TcpBuf Teil der Struktur UDT_TcpOut.

```
TYPE
        ARR_B_0_1460            :       ARRAY[0..1460] OF BYTE;
        ARR_B_0_100             :       ARRAY[0..100] OF BYTE;
END_TYPE

TYPE
        UDT_TcpBuf              :
        STRUCT
        Len                     :       int;
        RsvChr                  :       ARR_B_0_100;
        END_STRUCT;

        UDT_TcpOut              :
        STRUCT
        Len                     :       int;
        SndChr                  :       ARR_B_0_1460;
        TcpBuf                  :       UDT_TcpBuf;
        END_STRUCT;
END_TYPE
```

Text 4.2: Nutzung einer Struktur und eines Arrays in einer Struktur

Die Erstellung von Strukturen der Typen STRUCT und ARRAY sind nicht die einzige Möglichkeit, komplexere Datenstrukturen zu bilden. Zwar sind Strukturen zur Verwendung in allen POE vorgesehen, aber POE auch selbst sind zur Datenstrukturierung anwendbar.

```
TYPE
        UDT_TcpBuf              :
        STRUCT
        Len                     :       int;
        RsvChr                  :       String;
        END_STRUCT;
END_TYPE
```

Text 4.3: Struktur des Funktionsbausteins FB_1 (Text 4.4)

Dies trifft im Besonderen auf die Funktionsbausteine zu. Sie bilden POE wie Programme, müssen allerdings zuvor instanziert werden. Bei der Instanzierung wird der Funktionsbaustein einmal ausgeführt. Mit Hilfe der Instanz in Form einer Objektvariable sind Operationen auf die im Funktionsbaustein implementierten Variablen möglich. Voraussetzung für das Schreiben und Lesen von Strukturvariablen in Funktionsbausteinen ist die Deklaration der Variablen als VAR_IN_OUT. Bei einer Deklaration nur als VAR, VAR_IN oder VAR_OUT ist keine Operation bzw.

ausschließliches Schreiben eines Übergabe- oder Lesen des Rückgabeparameters machbar. Das Beispiel in Text 4.5 zeigt ein Programm *Main*. Durch Instanzierung wird der Funktionsbaustein *FB_1* instanziert und in der Objektvariablen *FB_1_1* gesichert. Über diese Objektvariable ist der lesende und schreibende Zugriff auf die Struktur *TcpBuf* möglich, da diese zuvor als VAR_IN_OUT deklariert wurde.

```
FUNCTION_BLOCK FB_1

VAR_IN_OUT
        TcpBuf                  :       UDT_TcpBuf;
END_VAR

TcpBuf.Len                      :=      20;
TcpBuf.RsvChr                   :=      'A';

END_FUNCTION_BLOCK
```

Text 4.4: Funktionsblock mit Datenstruktur

```
PROGRAM Main

VAR
        FB_1_1                  :       FB_1;
        MTcpBuf                 :       UDT_TcpBuf;
END_VAR

MTcpBuf.Len                     :=      20;
MTcpBuf.RsvChr                  :=      'B';
FB_1_1(
        TcpBuf                  :=      MTcpBuf);
        MTcpBuf                 :=      FB_1_1.TcpBuf;
END_PROGRAM
```

Text 4.5: Zugriff auf die Datenstruktur im Funktionsbaustein aus Text 4.4

Der dritte existierende POE-Typ, die Funktion, ist für die Datenstrukturierung nicht geeignet. Allerdings ist eine Anwendung der POE von diesem Typ alternativlos für die Implementierung von Fähigkeiten, die neben der strukturierten Datenhaltung für die Beschreibung von Objekten wichtig sind. Funktionen dienen der direkten Datenbearbeitung oder der Ausführung von mindestens einer Aktion. Die zu bearbeitenden Daten werden übergeben und als Rückgabewert nach Funktionsende bereitgestellt. Daten können jedoch nicht länger als einen SPS-Zyklus gehalten werden.

Ein einfaches Beispiel ist in Text 4.6 und Text 4.7 dargestellt. Das Programm *Main* ruft die Funktion *Func_1* auf und übergibt einen Übergabeparameter. Dieser wird durch die Funktion bearbeitet und als Rückgabewert in der Variable *InVar* gespeichert. Der Rückgabewert ist vom selben Datentyp wie der Übergabeparameter.

```
FUNCTION Func_1:BOOL

VAR_INPUT
        InVar                   :       BOOL;
END_VAR

        InVar                   :=      false;

END_FUNCTION
```

Text 4.6: Einfache Funktion zur Nutzung in anderen POE

```
PROGRAM Main

VAR
        MBuf                    :       String;
END_VAR

TcpBuf.RsvChr                   :=      Func_1(true);

END_PROGRAM
```

Text 4.7: Nutzung einer Funktion (Text 4.6) in einem Programm

Neben diesen für die Programmierung nach IEC 61131 typischen Merkmalen, ist ST auch durch einige eingeschränkte Fähigkeiten gekennzeichnet, die in der späteren Verwendung unbedingt Berücksichtigung finden müssen. Wurde die Steuerung der Variablen bislang nur durch die Schnittstellenvariablen, wie VAR_IN, VAR_OUT und VAR_IN_OUT, betrachtet, existieren nur zwei Möglichkeiten, um Einfluss auf die Sichtbarkeit der Variablen zu nehmen. Definiert man eine Variable lokal, d.h. möglichst in einer POE, ist diese auch nur dort nutzbar. Der Zugriff durch eine andere POE ist unterbunden und nur durch die vorgenannten Schnittstellenvariablen denkbar. Die Alternativ dazu kann die globale Definition sein, wodurch eine Variable in allen POE ansprechbar ist.

Innerhalb der Software spielt jedoch nicht nur der Austausch von Daten in Variablen eine Rolle. Auch die Übergabe von POE-Referenzen, speziell die von Funktionsbausteinen und Funktionen, ist eine gewichtiges Mittel zur Programmstrukturierung. Zur Zeit ist diese Funktionalität der IEC 61131 jedoch noch nicht in der Entwicklungsumgebung PCWORX implementiert worden. Dieser Umstand wird an einem späteren Punkt besonders betrachtet werden

4.3.2.3 Vergleich: Objektorientierung mit einer Hochsprache

Der vorher gehende Abschnitt befasste sich mit der Kurzbeschreibung der Programmiersprache ST zur direkten Programmierung der Phoenix-Steuerungen. Im Resultat steht eine Möglichkeit der Programmierung mit einer rein prozeduralen Programmiersprache zur Verfügung. Im Vergleich

sollen die bereits in Abschnitt 3.3 anhand der Hochsprache Java erläutert werden. Es wurden die vier Merkmale

- Datenabstraktion,
- Datenkapselung,
- Vererbung und
- Polymorphie

beschrieben. In der objektorientierten Programmiersprache Java finden diese Merkmale ihre Entsprechungen.

Eine Datenabstraktion findet Anwendung, wenn die Eigenschaften und Fähigkeiten eines Objektes nur allgemein beschrieben werden. Betrachtet man eine Windenergieanlage mit ihren Fähigkeiten *Start()* und *Stopp()* sowie der Eigenschaft *Bereit*, so müssen diese bei abstrakter Beschreibung nicht bis ins Detail bekannt sein, sondern können beispielsweise durch Methodenrümpfe bekannt gemacht werden. In Text 4.8 ist eine abstrakte Klasse *WEA* in Java dargestellt.

```
public class WEA
{
        privat          boolean         Bereit;
…
        void Stopp()
        {}

        void Start()
        {}
}
```

Text 4.8: Abstrakte Beschreibung der Eigenschaften und Fähigkeiten einer WEA in einer Java Class

Darüber hinaus wird in dieser abstrakten Klasse festgelegt, wie und ob die Daten sichtbar sind. Die Datenkapselung sorgt für die Eigenschaft *Bereit* dafür, dass diese nur innerhalb der Class *WEA* sichtbar ist, was hier durch das Schlüsselwort *privat* gewährleistet wird. Die Funktionen sind hingegen nach außen sichtbar und wären damit nach Instanzierung ansprechbar.

Ein Weg, um die abstrakte Klasse mit ihren Eigenschaften und Fähigkeiten nutzbar zu machen, ist eine Vererbung. In Text 4.9 erbt die Klasse *WEA_1* alle Eigenschaften und Fähigkeiten der Klasse *WEA* als abgeschlossenes Objekt. Es ist in der Klasse WEA_1 möglich, Informationen zu überschreiben oder zu ergänzen. Beispielsweise werden die Methodenrümpfe *Start()* und *Stopp()* erst durch Überlagerung mit einer gleichnamigen Funktion mit Anweisungen nutzbar. Zusätzlich werden die geerbten Eigenschaften und Fähigkeiten um die Funktion *Monitor()* und eine Eigenschaft *Betriebsstunden* ergänzt. Diese Realisierung entspricht dem letzten Merkmal der

Objektorientierung, der Polymorphie.

Mit Blick auf diesen Vergleich werden nun zwei Ansätze vorgebracht, die den konzeptionellen Unterschiede zwischen der objektorientierten IEC 61400-25 und der prozeduralen Programmiersprache ST zu überbrücken versuchen.

```
protected class WEA_1
extends WEA
{
        privat      boolean     Bereit;
        protected   int         Betriebsstunden     =     100;
…
        void Stopp()
        {
        ...Anweisungen...
        }

        void Start()
        {
        ...Anweisungen...
        }

        void Monitor()
        {
        ...Anweisungen...
        }
}
```

Text 4.9: Class WEA_1 erbt alle Eigenschaften und Fähigkeiten der Class WEA und erweitert diese

4.4 ST in Kombination mit einer objektorientierten Hochsprache

4.4.1 Systemarchitektur und Funktionsweise

Die Gegenüberstellung der Programmiersprache ST zur Programmierung und Steuerung des RFC und einer objektorientierten Hochsprache lassen fragen, ob eine Kombination der Fähigkeiten machbar ist. Dazu ist in Abbildung 4.8 (Variante A) eine mögliche Systemarchitektur aus drei Teilen bestehend dargestellt. Da die Voraussetzungen für die direkte Nutzung einer objektorientierten Hochsprache im RFC fehlen, ist die Verwendung einer zusätzlichen Hardware für einen Applikationsrechner, zzgl. eines geeigneten Betriebssystems, nicht zu vermeiden.

Der Übergang von und zur Feldebene wird durch den RFC im ersten Teil (*RFC*) gewährleistet. Alle Daten werden bidirektional zwischen den Feldbusteilnehmern und dem RFC ausgetauscht. Dem Datenaufkommen werden weitere Daten hinzugefügt, die aus dem RFC selbst akquiriert werden. Eine mit ST programmierte Software für den RFC hat die Aufgabe, diese Datenübermittlung von

und zur Feldebene abzuwickeln. Um die Verbindung zum zweiten Teil (*Applikationsrechner*) der Systemarchitektur herzustellen, kann eine der Schnittstellen genutzt werden, die sowohl mit ST als auch in einer Hochsprache standardmäßig einfach zu implementieren ist. Für das dafür erforderliche TCP/IP-Socket steht in ST als Funktionsbaustein und in vielen objektorientierten Hochsprachen als Klasse zur Verfügung. Damit ist eine Client-Server Verbindung realisierbar, die eine Datenübermittlung vom RFC zum Applikationsrechner umsetzt. Die Strukturierung der Daten sowie das Angebot der ACSI-Services ist Aufgabe der Applikationssoftware. Der Vorteil dieser Lösung liegt in der vereinfachten Implementierung des objektorientierten IEC 61400-25 Standards in einer gleichfalls objektorientierten Hochsprache. Mit Blick auf die Implementierung und eine zu erwartende Skalierung oder Wartung einer solchen Software, liegen die Vorzüge an dieser Stelle auf der Hand.

Schon im Abschnitt 3.4 wurde erläutert, dass die IEC 61400-25 nur Server-Funktionalitäten beschreibt. Die Schnittstelle zwischen Client und Server liegt aus logischer Sicht zwischen dem *Applikationsrechner* und dem Teil *Beobachtung*, auch wenn der *RFC* zur Komplettierung dieser Funktionalität zwingend notwendig ist. Für die Kommunikation zwischen dem Applikationsrechner und der Beobachtung ist eine weitere Client-Server Architektur unter Nutzung der TCP/IP Protokolle vorgesehen. Dies ermöglicht die Verwendung des MMS Kommunikationsmappings, nach der Art des Profils 3 in Abbildung 3.13.

In Abhängigkeit des dem TCP-Protokoll überlagerten Protokolls sind entsprechende Stacks vorzusehen. Für das vorgenannte MMS-Protokoll ist ein MMS Stack je für den Server des Applikationsrechners sowie für jeden Beobachtungsclient einzuplanen. Für die Kommunikation von und zum RFC ist ein solches Protokoll jedoch ungeeignet, das es bereits für den Zugriff auf objektorientierte Daten- und Dienstestrukturen entwickelt wurde. Hier würde ein einfaches Format seinen Zweck hinreichend erfüllen. Dementsprechend ist eine einfachere Programmierung der auch dafür erforderlichen Protokollstacks zu erwarten.

4.4.2 Bewertung der Vor- und Nachteile

Nach Beschreibung der Systemarchitektur ist ein zusammenfassender Blick auf die Vor- und Nachteile dieses Vorschlags angemessen.

Vorteile dieser Variante sind zuerst in der deutlich vereinfachten Implementation des sehr umfangreichen IEC 61400-25 Daten- und Datenaustauschmodells mit einer objektorientierten Hochsprache zu sehen. Es ergeben sich weitere Vorteile durch verfügbare Fähigkeiten einer Hoch-

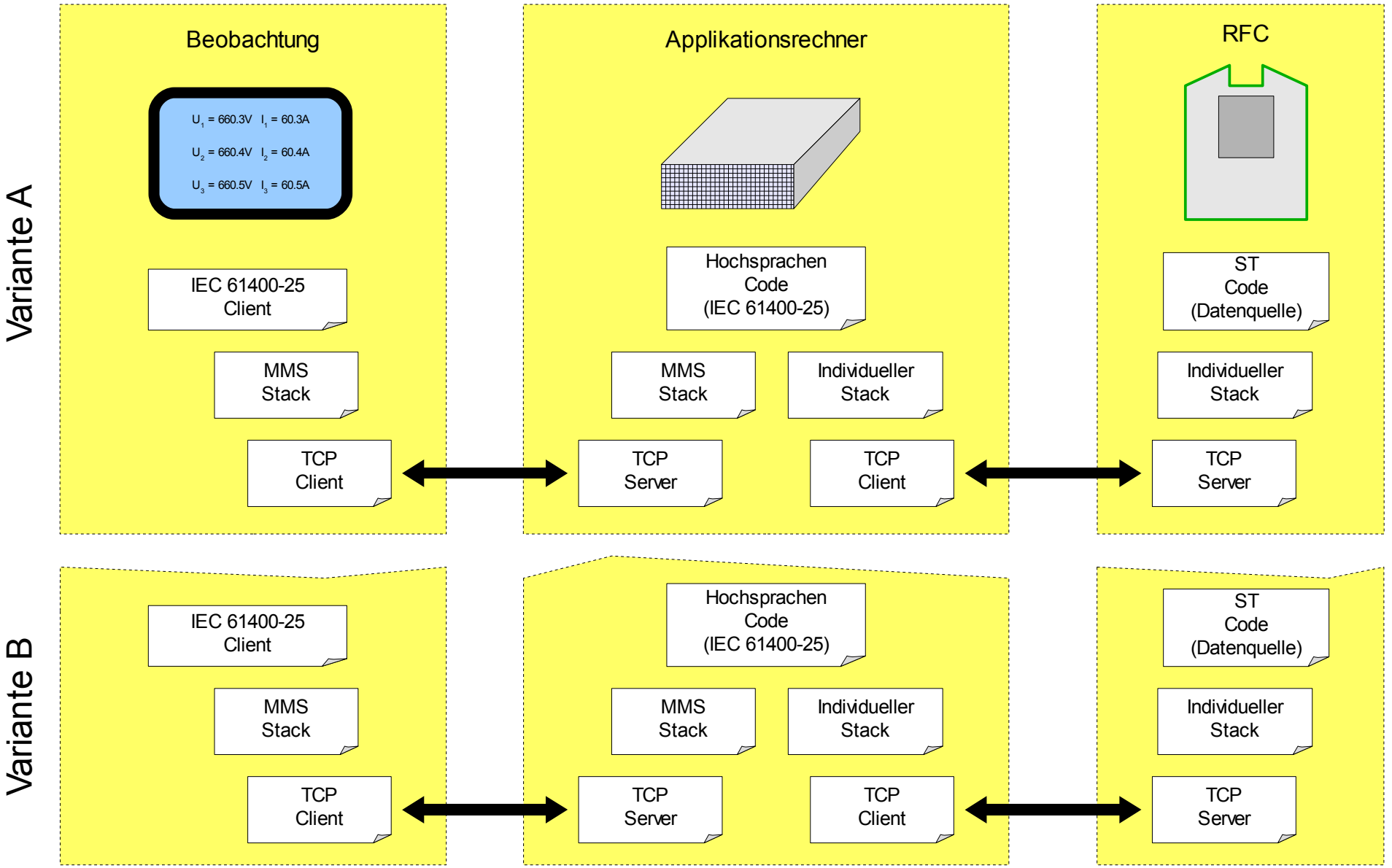

Abbildung 4.8: Darstellung der nötigen Systemarchitektur für den ersten Ansatz zur Lösungsfindung

sprache, die durch Programmiersprachen nach IEC 61131 derzeit nicht geleistet werden. Als Beispiel sei die dynamische Erstellung von Variablen und Klasseninstanzen genannt. Auf Seiten des RFC und seiner Programmierung in ST ist der Vorteil darin zu sehen, dass die komplexen Strukturen nicht umgesetzt werden und ggf. vorhandene ST-Projekte nur für die Kommunikation zum Applikationsrechner angepasst werden müssen. Weitere Vorteile ergeben sich aus der später diskutierten alternativen Systemarchitektur.

Dem gegenüber hat diese Lösungsvariante jedoch auch eine Reihe von Nachteilen. Als erstes fällt der zusätzliche Bedarf an der Hardware für den Applikationsrechner ins Auge. Da die IEC 61400-25 für Windenergieanlagen und Windparks gedacht ist, sind entsprechende Ansprüche bezüglich der Widerstandsfähigkeit dieses Rechners bzw. seiner Behausung zu stellen und letztlich auch zu bezahlen. Ein zweites Manko stellt die ausschließliche Nutzung der TCP/IP Protokolle dar. Der Vorteil der einfach zu implementierenden Kommunikationskomponenten erkauft man sich allerdings mit dem Verzicht auf Echtzeiteigenschaften, da das TCP-Protokoll nicht deterministisch ist und die Übertragungsgeschwindigkeit von der Menge der zu übertragenden Daten abhängt. Zusätzlich weist dieses Systemarchitektur vom Beobachtungsrechner bis zum RFC gleich zwei Client-Server Anwendungen auf. In einem nächsten Schritt soll versucht werden, das Verhältnis der Vor- zu den Nachteilen günstig zu beeinflussen.

4.4.3 Optimierung der Systemarchitektur

Die Nachteile des vorgeschlagenen ersten Lösungsansatzes resultieren u.a. aus der umständlichen Systemarchitektur. Es besteht allerdings ein Spielraum, durch dessen Nutzung wesentliche Verbesserungen erreicht werden können (Abbildung 4.8, Variante B).

Die IEC 61400-25 definiert die Serverfunktion nicht allein in einem Steuerungsrechner wie dem RFC allein. Es auch möglich, diese Funktion beispielsweise in Proxy- oder Applikationsrechnern zu konzentrieren. Im Normalfall kann ein RFC zur Steuerung einer Windenergieanlage angenommen werden. Daher eröffnet sich die Möglichkeit, einen Applikationsrechner für eine Gruppe von Windenergieanlagen vorzusehen. Das würde jedoch bedeuten, dass entsprechend viele Clients oder ein entsprechend aufwendiger Client auf dem Applikationsrechner für die Kommunikation zu den RFCs vorgehalten werden müssten. Eine wesentliche Optimierung wird hier erreicht, in dem die Serverfunktion allein im Applikationsrechner realisiert wird. Der zu implementierende Multithread-Server hätte die Aufgabe, die Kommunikation zu den Clients auf dem RFC ebenso abzuwickeln, wie zu den Beobachtungsclients. Für eine konkrete Umsetzung spielen alle Protokollstacks eine

bleibende Rolle.

Nimmt der Server nur Verbindungen auf einer Ethernet-Schnittstelle an, gilt zusätzlich festzulegen, wie die Unterscheidung zwischen den Protokollen erfolgen soll. Es wäre beispielsweise möglich, diese beim Verbindungsaufbau durch den Client durch eine Anfangssequenz zu unterscheiden, wenn dies im Umfang eines Übertragungsprotokolls liegt. Auch wäre eine Unterscheidung anhand der IP-Adresse möglich. Lösungen dieser Art haben sind durch geringe Änderungen der Implementierung nicht vor Fehlern geschützt. Ein praktikablerer und transparenterer Ansatz ist der Betrieb auf zwei Ports. Verbindungen der Clients werden je nach Protokoll auf den jeweiligen Port verbunden. Bei Verbindungsannahme durch den Server wird der jeweilige Protokollstack mit den Telegramm-inhalten assoziiert.

Abschließend ist festzustellen, dass dieser Lösungsansatz zwar möglich ist, aufgrund zweier gewichtiger Gründe nicht die erste Wahl für die Umsetzung darstellt. Phoenix beliefert Windenergieanlagenhersteller mit Automationskomponenten wie dem RFC. Zusätzliche Applikationsrechner gehören normalerweise nicht dazu und sind oft in keiner Planung vorgesehen. Zudem ist es wünschenswerter, eine Abbildung der IEC 61400-25 direkt in ST zu realisieren, um diese Funktionalität direkt im RFC anbieten zu können. Nachfolgend wird diesbezüglich ein Ansatz untersucht.

4.5 Ansatz 2: Modelltransformation nach ST

4.5.1 Grundüberlegung für eine Modelltransformation

Um dem gesetzten Ziel der direkten Implementation eines objektorientierten Standards gerecht zu werden, muss die Frage nach der Art und Weise der Implementierung gestellt werden. Werden zwei Modelle unter Beibehaltung von Eigenschaften und Fähigkeiten ineinander umgewandelt, spricht man von einer Modelltransformation. Eine Transformation erfolgt nach Regeln. Der Absicht folgend, objektorientierte Modelle der IEC 61400-25 zu übertragen, sind diese Regeln aufzustellen.

Wie in Abbildung 4.9 zu erkennen ist, sind die objektorientierten Modelle für die Kommunikation sichtbar. Die Trennung durch Schichten, wie etwa das Mapping, verhindert die direkte Sichtbarkeit. Damit ergibt sich ein gewisser Spielraum bei der Wahl der Systemarchitektur. Zur Wahl stehen nahezu direkte Entsprechungen, wenn die IEC 61131 Implementierung eines Herstellers dies zulässt oder oder eine Nachbildung in Fähigkeiten und Eigenschaften, die keine direkte Entsprechung ist.

Der Vorteil der direkten oder weitgehend direkten Entsprechung kann darin gesehen werden, meist komplexe objektorientierte Modelle auch nach der Implementation (z.B. in ST) als objektorientiert erkennbar sind. Zudem sind die o.g. Transformationsregeln einfach gehalten. Sie beschreiben im Wesentlichen Ersetzungen. Beispielsweise könnte beschrieben werden, wie eine Klasse in ST unter Beibehaltung von Eigenschaften und Fähigkeiten zu erzeugen ist. Dabei sei auf die Gegenüberstellung in Kapitel 4.3.2 hingewiesen. Ein potentieller Nachteil ist die möglicherweise anstehende Übertragbarkeit des Codes auf herstellerfremde Automatisierungssysteme, deren Implementierung der IEC 61131 von Erstellersystem abweicht.

Dem gegenüber steht die Modelltransformation, wenn die Eigenschaften und Fähigkeiten nicht direkt nachgebildet werden, mit umfangreicheren Regeln. Die nachträgliche Lesbarkeit ist nicht mehr gegeben und stellt einen Nachteil bei Änderungen dar. Allerdings stellt dieser Weg der Modelltransformation für manche Anwendungen den einzig machbaren Weg dar.

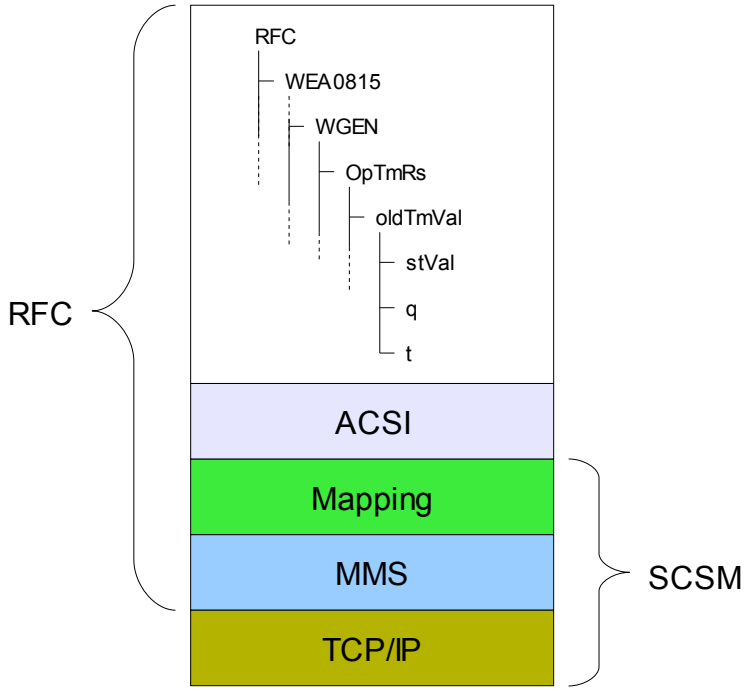

Abbildung 4.9: Darstellung der Schichten in einer möglichen Systemarchitektur

4.5.2 Kurzbeispiel einer direkten Implementation in ST

Ein möglicher Weg zur Integration der Objektorientierung in ST wird in [HPVHB05] („Möglichkeiten der Darstellung von Zustandsautomaten in der IEC 61131-3“) beschrieben. Dort wird unter Verwendung von Automationskomponenten der Fa. Beckhoff u.a. eine Modelltransformation vorgenommen. Essentieller Teil der Objektorientierung ist die Darstellung in

sich abgeschlossener Objekte. Hierfür werden nach [HPVHB05] instanzierbare Funktionsbausteine genutzt. Mit einer Instanz eines Funktionsbausteins ist ein Zugriff auf die Eigenschaften und Fähigkeiten (Funktionen) möglich, indem diese in Strukturen implementiert werden (Abbildung 4.10).

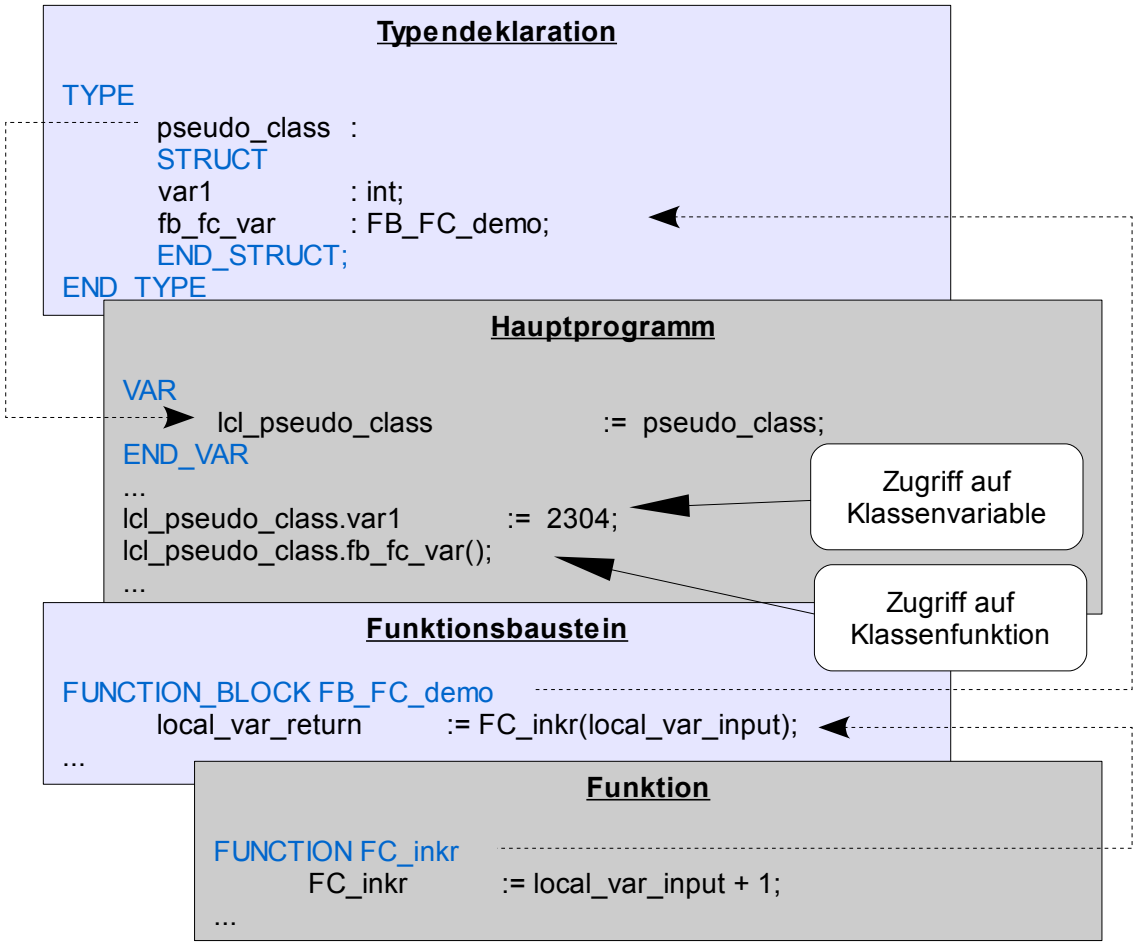

Abbildung 4.10: Beispiel einer direkten Implementierung in ST nach [HPVHB05]

Speziell die Einschränkung der Phoenix Steuerungssysteme, keine Funktionen und Instanzen von Funktionsbausteinen als Referenz übergeben oder in Strukturen verwenden zu können, verhindern die direkte Anwendung der hier vorgestellten Regeln. Unter dem Gesichtspunkt, dass Eigenschaften und Funktionen nach IEC 61400-25 durch die Kommunikation sichtbar sein sollen, wird im nächsten Schritt eine endgültige Lösungsvariante vorgestellt.

4.5.3 Allgemeines Systemverhalten

Unter der vorgenannten Prämisse, sämtliche Modelle für die Kommunikation sichtbar zu machen, wird der Ansatz zur Implementierung unter Beibehaltung der Eigenschaften und Fähigkeiten

verfolgt. Es werden an den Server gerichtete Nachrichten verarbeitet und ein Resultat an den Client zurück geliefert. Jedoch wird die tatsächliche serverseitige Implementierung in ST vor diesem verborgen. Der Sachverhalt einer Client-Server Kommunikation ist im Aktivitätsdiagramm in Abbildung 4.11 sehr allgemein dargestellt.

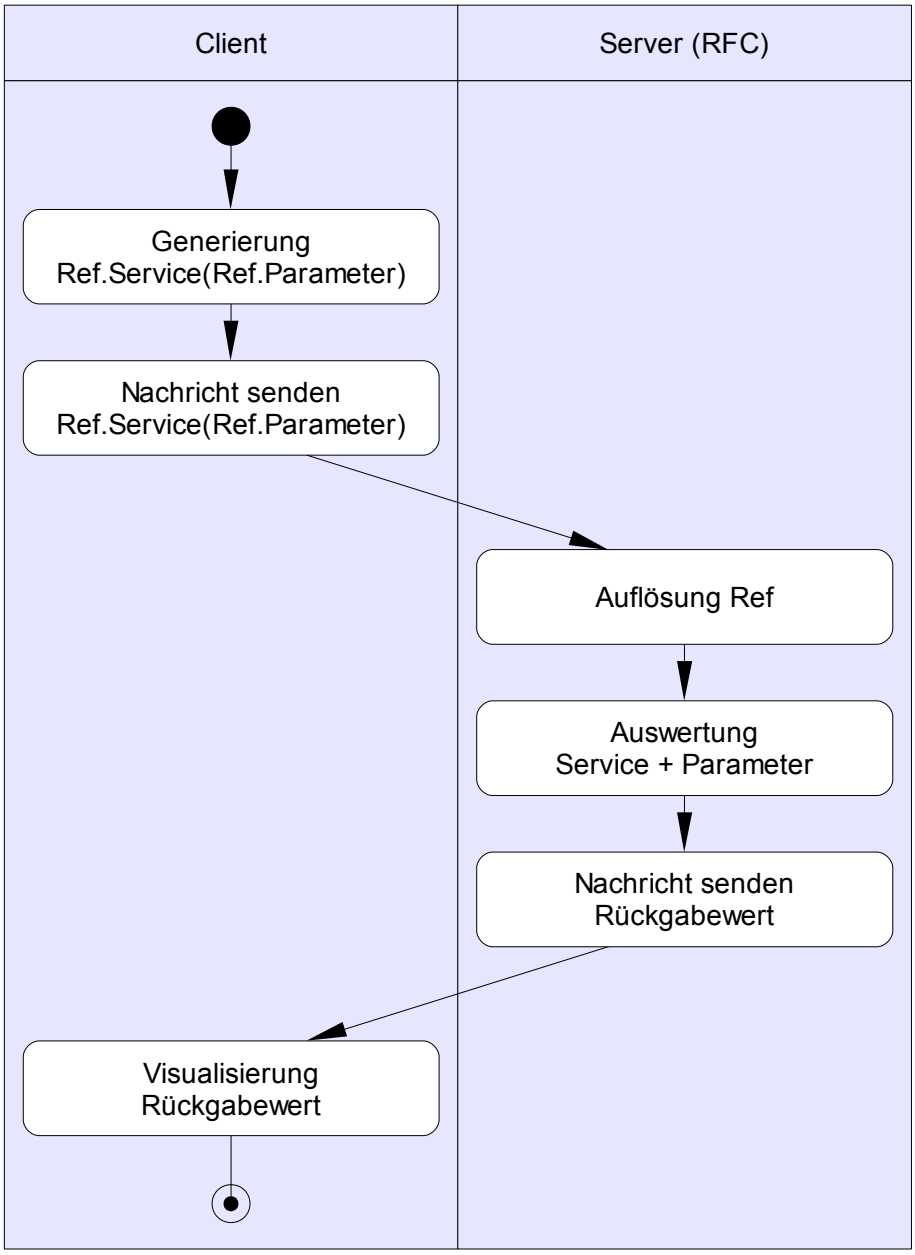

Abbildung 4.11: Aktivitätsdiagramm für die Vorgänge

Um die Erläuterungen in dieser Richtung fortzusetzen, sind einige Festlegungen zu treffen. Da die Umsetzung unter Berücksichtigung von [IEC400-25-4] nicht teil dieser Arbeit ist, wird ein einfaches Nachrichtenformat gewählt, IEC 61400-25 Praxis so nicht verwendet wird, aber für die Betrachtungen und Erklärungen zur Teilimplementierung in ST aber ausreicht. Das

Nachrichtenformat entspricht der Form

RefS.Service([RefP1.Parameter][,RefP2.Parameter]...)

und orientiert sich an einer objektorientierten Zugriff. Das Mapping und die Implementation eines Übertragungsprotokolls wird damit komplett ausgeblendet und bleibt damit ein Kandidat für die Weiterentwicklung. Die Referenzen (*RefS* und *RefP**) entsprechen dem jeweiligen Pfad zum angesprochenen Objekt, z.B. einer LogicalNode Class. Die Objekte sind mit ihren Eigenschaften (Variablen) und Fähigkeiten (ACSI-Services) in Klassen organisiert. Die Services sind an Klassen gebunden und nur für einen Klassentypus gleich, wenn sie nicht abgewählt wurden. Wie in der Objektorientierung üblich werden Services und übergebene Parameter ohne Referenz direkt im aktuell referenzierten Objekt gesucht. Es werden des weiteren nur absolute Referenzen verwendet.

Die Algorithmen der Teilimplementierung befassen sich zusammengefasst mit

- dem ACSI,
- dem Protokollstack,
- der Umsetzung eines Datenmodells und
- der Umsetzung serverinterner Aktivitäten, wie z.B. Trigger und Functional Constraints.

Sie sind im Funktionsbaustein IEC61400_25_V1_00 gekapselt. Der Quellcode findet sich im Anhang ab Seite 84.

4.5.4 Nachrichtenaustausch und Servicebehandlung

4.5.4.1 Nachrichtenaustausch

Die Umsetzung des Nachrichtenaustausches wird in der Teilimplementierung über die TCP/IP Protokolle unter Verwendung des o.g. Nachrichtenformates abgewickelt. Für die Implementierung kann clientseitig ein einfaches Terminalprogramm verwendet. Üblicherweise sind diese Programme im Falle einer fertiggestellten Implementierung sehr umfangreich stellen eine vollständige HMI dar. Für die Entwicklung dieser Basisalgorithmen jedoch ist eine einfache Variante ausreichend, auch um den Nachrichtenaustausch für Entwicklungszwecke demonstrativ und transparenter zu machen.

Fast alle komplexen Funktionalitäten der Programmierung nach IEC 61131 stehen in der Entwicklungsumgebung PCWORX als Funktionsbausteine zur Verfügung. Die Implementierung des Nachrichtenaustausches in ST macht deshalb hauptsächlich von den Funktionsbausteinen IP_CONNECT, IP_USEND und IP_URCV Gebrauch. Mit dem Aufruf von IP_CONNECT und dem Parameter *passiv* wird der Baustein in die Lage versetzt, eingehende Verbindungen anzunehmen und somit als Server zu fungieren. Gemäß dieser Funktion ist es nur gestattet, auf Anfragen des Clients zu antworten, nicht jedoch eigenständige Verbindungen zum Client aufzubauen.

Diese Teilimplementierung nutzt weiterhin die Option der Selektierung eines einzelnen Kommunikationspartners durch Angabe der IP-Adresse.

Um das TCP Protokoll verwenden zu können, kommen die Bausteine IP_USEND und IP_URCV zur Anwendung. Sie realisieren den Versand und Empfang der Nachrichten mit einer maximalen Größe von 1460 Bytes. Dies entspricht der maximalen Nutzdatenmenge, die pro TCP-Paket übertragen werden können. Die Verwendung der Bausteine IP_USEND und IP_URCV setzt eine bestehende Verbindung mit IP_CONNECT voraus. Die Bausteine werden durch Übergabe einer ID miteinander referenziert, da auch mehrere IP-Bausteine in einer POE aktiv sein können. Programmseitig werden die Bausteine mit je einen Bytestream für die Ein- und Ausgabe der Daten bedient. Für die Umwandlung in andere Datentypen stehen entsprechende Funktionen und Funktionsbausteine zur Verfügung.

4.5.4.2 Servicebehandlung

Für eingehende Nachrichten wird grundsätzlich das eingeführte Format RefS.Service([RefP.Parameter][,RefP.Parameter]...) erwartet. Um das Format auswerten zu können, ist die Zerlegung in seine Bestandteile

- Referenz des Services (*RefS*),
- Referenz des Parameters (*RefP**),
- Service bzw. Servicenamen (*Service*) und
- Parameter bzw. Parameternamen (*Parameter**)

notwendig, wobei hier nicht kein, ein oder mehr als ein kein Parameter übergeben wird. Die Objektreferenzen *RefS* und *RefP** müssen nicht identisch sein. Dazu werden die eingehenden Daten aus dem Bytestream des Funktionsbausteins IP_URCV grundsätzlich in einen String umgewandelt.

Dies ist in der Tatsache begründet, dass die Weiterverwendung auf Basis des Datentyps String geschieht.

Die Bearbeitung der Serviceanfrage wird durch die jeweilige Parametertabelle vorgegeben, wie sie in Tabelle 3.6 dargestellt ist. Es wird mit der Prüfung begonnen, ob ein Service für ein referenziertes Datenobjekt verfügbar und zulässig ist. Führt ein Schritt zu einem negativen Ergebnis, wird eine Fehlermeldung an den Client zurück gesandt (*Response-*). Fällt die Prüfung positive aus, kann eine entsprechende Ausfürhung des Services stattfinden und eine Antwort gesendet werden (*Response+*), sofern dies vorgesehen ist. Für die Form der Meldungen *Response-* und *Response+* macht die IEC Parametertabelle keine genauen Angaben. Es wird deshalb ein für den Benutzer verständliches Format gewählt.

4.5.5 Umsetzung des Datenmodells

4.5.5.1 Identifizierung von Datenobjekten im Datenmodell

Das ausschnittsweise Datenmodell in [IEC400-25-4] Figure A.2 dient für die folgenden Erläuterungen zur Implementierung des Datenmodells in ST als Basis. Um die Nachvollziehbarkeit zu erleichtern, ist es in Abbildung 4.12 nochmals mit Ergänzungen und konkreten Quellenbezug dargestellt.

Das Datenmodell stellt sich als Baumstruktur dar. Jedes Element verkörpert ein Datenobjekt eines Typs nach IEC 61400-25, die dort als Class bzw. Klasse definiert sind. Um den Zusammenhang der Baumelemente untereinander herzustellen, halten Datenobjekte Instanzen untergeordneter Datenobjekte. Jedes Datenobjekt bzw. jede Klasse muss eindeutig über Name und Objektreferenz zu identifizieren sein. Datenobjekte mit gleichen Namen und gleicher Objektreferenzen sind nicht erlaubt, da sie sie nicht eindeutig zu identifizieren sind. Dieser Sachverhalt spielt bei der Objektbenennung nach IEC 61400-25 eine wichtige Rolle. Betrachtet man den Aufbau des o.g. beispielhaften Datenmodells, so sind gleiche Benennungen der Datenobjekte in der ersten bis dritten Ebene möglich. Demzufolge sind Physical Devices, Logical Devices und Logical Nodes individuelle Namen zuzuweisen, um Doppelungen zu vermeiden. Für Datenobjekte unterhalb der ersten Logical Nodes sind deren Eigennamen möglich. Doppelte Datenobjekte innerhalb der Logical Nodes wurden hingegen in keinem Dokument zur IEC 61400-25 aufgefunden. Der Grund für diese „Zweiteilung“ des Datenmodells bezüglich der Objektbezeichnung liegt darin, dass Datenstrukturen, aus denen Logical Nodes bestehen, fest vorgegeben sind und somit den unteren

Teil des Datenmodells ausmachen. Hingegen müssen Physical und Logical Devices sowie die Anzahl der Logical Nodes nach den Objekten der realen Welt ausgewählt werden, die abzubilden sind, und demnach individuell benannt werden.

Datenobjekte, die Teil eines Logical Nodes sind und Logical Nodes selbst, besitzen jedoch das Attribut *M/O*, das die An- und Abwahl eigener Datenobjekte erlaubt (z.B. Tabelle 3.2, Tabelle 3.4). So können sich Datenstrukturen, die unterhalb von Logical Nodes existieren trotz gleichen Namens, durch das Vorhandensein oder Fehlen eines oder mehrerer Felder, voneinander unterscheiden. Die eindeutige Identifikation zweier solcher Datenobjekte ist nur in Kombination mit zwei ungleichen Objektreferenzen möglich.

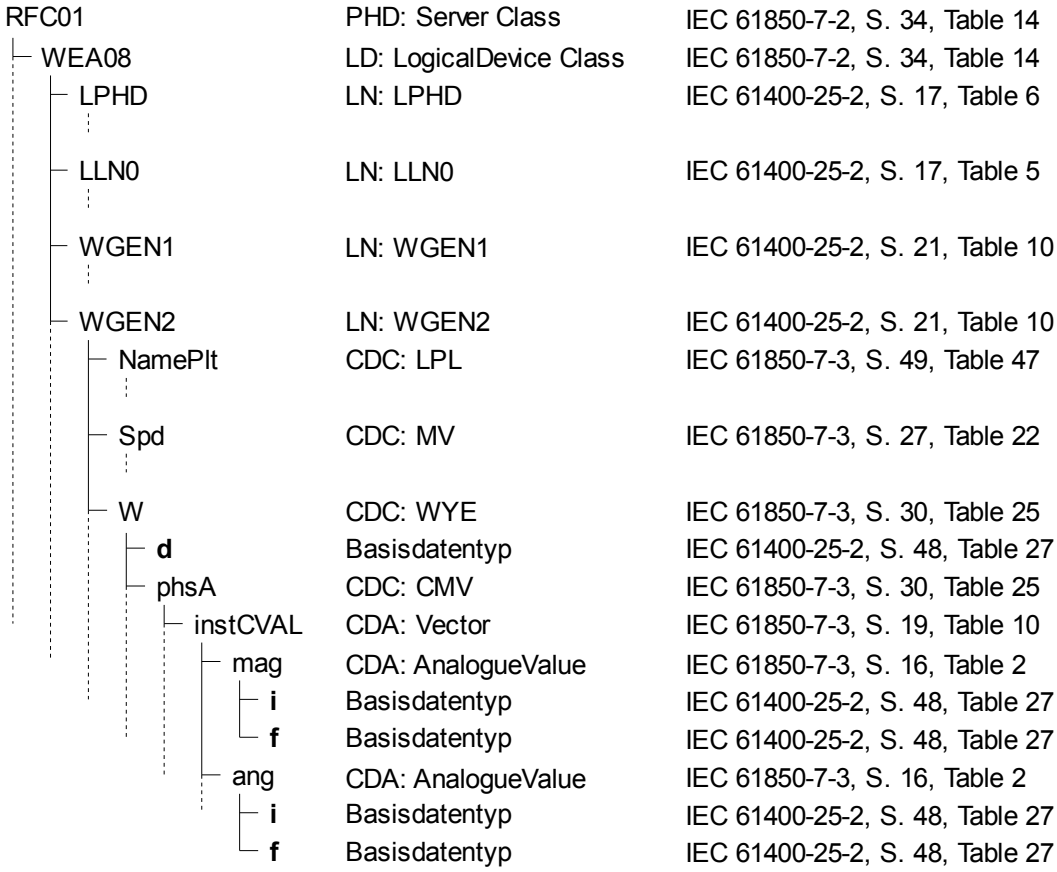

Abbildung 4.12: Beispiel zur Implementierung eines Datenmodells

In Abbildung 4.12 finden sich die Datenobjekte Physical Device und Logical Device, die den RFC (*RFC01*) als Serverrechner und eine angenommene Windenergieanlage *WEA08* repräsentieren, als höchste Elemente. Dem untergeordnet sind die beiden Logical Nodes *LPHD* und *LLN0* zur Beschreibung von RFC01 und WEA08 sowie zwei Logical Nodes WGEN1 und WGEN2 zur Beschreibung der Generatorinformationen. Das Auftreten zweier LN dieses Typs trifft eine Aussage über die Anzahl der Generatoren in dieser Windenergieanlage (An dieser Stelle muss angemerkt

werden, dass Windeneergieanlagen typischerweise nur einen Generator aufweisen. Bei Auftreten zweier Generatoren ist eher an eine kombinierte Bauform gedacht, um z.B. eine Synchronmaschine mit einen permanent erregten Generator zur Bereitstellung der Erregerspannung bei Netzausfall zu ergänzen.).

Die eindeutige Benennung ist bei der Herstellung der Zusammenhänge im Datenmodell essentiell. Betrachtet man die Datenobjekte vom Logical Node und Common Data Class, so stehen die Eigenschaften *Name* (Benennung) und *Type* (Datentyp) zur Verfügung. Über diese Eigenschaften wird die Verkettung der Datenobjekte untereinander realisiert. Die im Attribut *Name* befindliche Benennung verweist auf ein untergeordnetes Datenobjekt eines Datentyps. Datenobjekte vom Typ Physical Device und Logical Device bringen diese Eigenschaften nicht mit ins Datenmodell ein. Sie sind per Definition nur Container mit linearen Listen, in denen die untergeordneten Datenobjekte gespeichert sind. So wird der Logical Node *WGEN2* in dem behandelten Beispieldatenmodell auch mit einer Datenstruktur *WGEN2* assoziiert, die durch An- und Abwahl untergeordneter Datenobjekte aus dem Logical Node WGEN hervorgeht. Für die nachstehende Implementation gelten *WGEN2* wie auch *WGEN1* somit als neue und ungleiche Objektdatentypen.

4.5.5.2 Vergleich der Datentypen IEC 61400-25 vs. Datentypen PCWORX

Die Struktur des Datenmodells setzt sich, wie erläutert, aus einer Vielzahl von aufwendigen Objektdatentypen zusammen, wodurch sich eine komplexe Datenstruktur ergibt. Jedoch passiert der Zugriff letztlich auf Daten auf Basisdatentypen ([IEC400-25-2], S. 48, Table 27), die die unterste Ebene des Datenmodells bilden. Für die Implementation ist ein Vergleich der Basisdatentypen mit denen der IEC 61131 notwendig, um eine korrekte Abbildung zu gewährleisten.

In Tabelle 4.1 werden die Datentypen der IEC 61400-25 denen der IEC 61131 Implementierung der PCWORX Entwicklungsumgebung gegenübergestellt. Für nahezu alle Basisdatentypen existieren äquivalente Entsprechungen. Für die verbleibenden Datentypen ohne passende Datentypen nach IEC 61131 gibt es neben der Möglichkeit, Datentypen in begrenzten Rahmen neu zu definieren, auch die Kompromisslösung, eine eingeschränkte Entsprechung zu akzeptieren. In einer solchen Situation könnte ein Datentyp mit geringerem Wertebereich akzeptabel sein. Dieses Vorgehen wäre jedoch nicht konform und ist in der IEC 61400-25 nicht vorgesehen.

Neben den in der Tabelle 4.1 aufgeführten Datentypen, sind in PCWORX weitere Datentypen verfügbar (Tabelle 4.2, Tabelle 4.3). Im Zusammenhang mit der Abbildung des IEC 61400-25 Datenmodells finden sie jedoch keine direkte Anwendung.

PCWORX				IEC 61400-25			
Bez.	**Datentyp**	**Bits**	**Anmerkung**	**Bez.**	**Datentyp**	**Bits**	**Anmerkung**
SINT	8 Bit Integer mit Vorz.	8	-128 bis 127	INT8	8 Bit Integer mit Vorz.	8	-128 bis 127
INT	Integer mit Vorz.	16	-32.768 bis 32.767	INT16	16 Bit Integer mit Vorz.	16	-32.768 bis 32.767
DINT	Double Integer mit Vorz.	32	-2^{31} bis $2^{31}-1$	INT24	24 Bit Integer mit Vorz.	24	-83.88.608 bis 83.88.607
				INT32	32 Bit Integer mit Vorz.	32	-2^{31} bis $2^{31}-1$
LINT*	64 Bit Integer mit Vorz.	64	-2^{63} bis $2^{63}-1$	INT64	64 Bit Integer mit Vorz.	64	-2^{63} bis $2^{63}-1$
Ohne direkte Entsprechnung				INT128	128 Bit Integer mit Vorz.	128	-2^{127} bis $2^{127}-1$
USINT	8 Bit Integer ohne Vorz.	8	0 bis 255	INT8U	8 Bit Integer ohne Vorz.	8	0 bis 255
UINT	Integer ohne Vorz.	16	0 bis $2^{16}-1$	INT16U	Integer ohne Vorz.	16	0 bis $2^{16}-1$
UDINT	Double Integer ohne Vorz.	32	0 bis $2^{32}-1$	INT24U	Double Integer ohne Vorz.	24	0 bis $2^{24}-1$
				INT32U	32 Bit Integer ohne Vorz.	32	0 bis $2^{32}-1$
ULINT*	64 Bit Integer ohne Vorz.	64	0 bis $2^{64}-1$	Ohne direkte Entsprechnung			
REAL	32 Bit Gleitkommawert	32	$\pm 1{,}5\cdot 10^{-45}$ bis $\pm 3{,}4\cdot 10^{38}$	FLOAT32	32 Bit Gleitkommawert	32	$\pm 1{,}2\cdot 10^{-38}$ bis $\pm 3{,}4\cdot 10^{38}$ (IEEE 754,[WUES06])
LREAL*	64 Bit Gleitkommawert	64	$\pm 2{,}2\cdot 10^{-308}$ bis $\pm 1{,}8\cdot 10^{308}$	FLOAT64	64 Bit Gleitkommawert	64	$\pm 2{,}2\cdot 10^{-308}$ bis $\pm 1{,}8\cdot 10^{308}$ (IEEE 754,[WUES06])
STRING	Zeichenfolge	80 Byte		OCTETT STRING64	Oktettstring	64	
Ohne direkte Entsprechnung				ENUMERATED		k.A.	
Ohne direkte Entsprechnung				CODED ENUM		k.A.	
STRING (255)	Zeichenfolge	255 Byte	(Eingeschränkte Entsprechnung)	VISIBLE STRING255		k.A.	ASCII Zeichenkette
STRING	Zeichenfolge	80 Byte	(Eingeschränkte Entsprechnung)	UNICODE STRING		k.A.	Unicode Zeichenkette
BOOL	Bool	1	0 bis 1_{HEX}bzw. False/True	BOOLEAN	Bool	1	False/True

Tabelle 4.1: Entsprechungen der in [IEC400-25-2] für die IEC 61400-25 definierten Datentypen in PCWORX in [PHOE08]

Mit „*“ gekennzeichnete Datentypen sind noch nicht implementiert. Im Falle von LREAL wird angenommen, dass der Wertebereich IEEE754 entspricht.

Schlüsselwort	Datentyp	Größe in Bits	Wertebereich
BYTE	Bitstring 8	8	0 bis FF_{HEX}
WORD	Bitstring 16	16	0 bis $FFFF_{HEX}$
DWORD	Bitstring 32	32	0 bis FFFF $FFFF_{HEX}$
LWORD	Bitstring 64	64	0 bis FFFF FFFF FFFF $FFFF_{HEX}$

Tabelle 4.2: In PC WORX realisierte bitbasierte Datentypen nach IEC 61131 ([PHOE08])

Schlüsselwort	Datentyp	Größe	Wertebereich
TIME	Zeitdauer	32 Bit	0 bis 4.294.967.295ms
DATE	Datum		
TOD	Tageszeit		
DT	Datum und Zeit		

Tabelle 4.3: In PC WORX realisierte erweiterte Datentypen ([PHOE08])

4.5.5.3 Nachbildung des Datenmodells mit Strukturen

Mit der Klärung grundlegender Zusammenhänge des Datenmodells, kann die Umsetzung in ST begonnen werden. Der Aufbau des Datenmodell erfolgt statisch. Die Datenstruktur wird nach Festlegung in ST-Code programmiert und kann nicht mehr geändert. Auf dieser Datenbasis werden alle anderen Aktivitäten ausgeführt. Auch für die Auswahl einiger Daten mit einem besonderen Informationsschwerpunkt sind keine Änderungen an der Datenstruktur nötig. Datasets können während des Betriebes erzeugt werden, um Reports oder Logs zu generieren.

Die Eigenschaften der nachzubildenen Datenobjekte, die in der IEC 61400-25 zumeist als xyz Class bezeichnet werden, werden durch die zugehörigen Tabellen in [IEC850-7-2], [IEC400-25-2] und [IEC400-25-3] vorgegeben. In Vorgriff auf den Abschnitt 4.5.6 sei ergänzt, dass weitere Metainformationen zusätzlich für ein Datenobjekt hinzuzufügen sind. Für die Definition von Datenstrukturen existieren hier zwei Wege: Implementation durch Funktionsbausteine oder Strukturen (STRUCT). Bereits im Abschnitt 4.3.2.2 wurde auf die Optionen der Programmiersprache ST zum Aufbau einer Datenstruktur eingegangen, wie sie hier nun gebraucht wird. Für die Abbildung der Datenobjekte nach der vorliegenden Norm wäre eine Möglichkeit wünschenswert, die Eigenschaften und Fähigkeiten abbilden kann.

Funktionsbausteine sind primär für die Speicherung von komplexen Algorithmen in wiederverwendbaren Modulen gedacht. Die IEC 61131 Implementierung in PCWORX sieht jedoch keine Möglichkeit vor, Fähigkeiten in Form von Funktionen, zu speichern. Damit gelingt mit einer

Instanz eines Funktionsbausteins auch nur der Zugriff auf Datenstrukturen. Daraus ergibt sich kein Nachteil gegenüber dem Aufbau von Datenstrukturen mit selbst definierten Datentypen unter Anwendung der STRUCT Anweisung. Es ist im Allgemeinen üblich, Datenstrukturen in ST auf diesem Weg zu realisieren, die auch ineinander verschachtelt sein können. Die Verwendung dieser Methode erscheint hier als prädestiniert. Zudem lassen sich Datenstrukturen in PCWORX auf einem Arbeitsblatt konzentrieren, was die Arbeiten an komplexen Datenmodellen erleichtert. Aufgrund der bereits im Abschnitt 4.5.5.1 erläuterten Zusammenhänge kann der Aufbau des Datenmodells nun beginnen.

Der Aufbau der selbst definierten Strukturen des Datenmodells aus Abbildung 4.12 beginnt in der untersten Ebene, da Definitionen von STRUCT Anweisungen, die sich aus anderen Anweisungen dieser Art zusammensetzen, nur bereits bekannte Strukturen akzeptieren. Bei der Wahl der Datentypen ist die Gegenüberstellung aus Tabelle 4.1 zu beachten. Die unterste Struktur für das CDA *UDT_AnalogueValue* ist in Text 4.10 dargestellt. Die Namenserweiterung *UDT* (User Data Typ) erfolgt aufgrund einer hausinternen Programmiernorm der Fa. Phoenix. Da die Attribute i und f des CDA AnalogueValue nicht abgewählt werden können, da sie als „Mandatory“ gekennzeichnet sind, benötigt der Datentyp keine weitere Namensergänzung in Erwartung neuer CDA-Typen notwendig. Dies trifft auch für andere Objektdatentypen zu, die keine „Optional“ Attribute aufweisen, als nur mit „Mandatory“ gekennzeichnete Attribute aufweisen.

```
TYPE
        UDT_AnalogueValue       :
        STRUCT
        i                       :       int;
        f                       :       real;
        END_STRUCT;

        UDT_Vector_001          :
        STRUCT
        mag                     :       UDT_AnalogueValue;
        ang                     :       UDT_AnalogueValue;
        END_STRUCT;

        UDT_CMV_001             :
        STRUCT
        instCVAL                :       UDT_Vector_001;
        END_STRUCT;
END_TYPE
```

Text 4.10: Selbst definierte Datenstruktur CDA *UDT_AnalogueValue*, *UDT_Vector_001* und *UDT_CMV_001*

Das Datenobjekt der nächsthöheren Ebene, vom Typ Vector, hingegen besitzt zwei Attribute *mag* (Mandatory) und *ang* (Optional). Demnach sind zwei Vector Objektdatentypen möglich, unter Berücksichtigung des untergeordneten Objektdatentyps AnalogueValue, der sich nicht ändern kann.

Um eine Unterscheidung in solchen Fällen sicherzustellen, wird dem Objektdatentyp ein durchlaufender Identifier angefügt (siehe Text 4.10). Aus Sicht der IEC 61400-25 handelt es sich zwar um einen Objektdatentyp, die Umsetzung in ST erfordert jedoch die Definition zweier Datentypen, hier also UDT_Vector_001 und UDT_Vector_002 (Abbildung 4.13). Auf ein tiefer gehendes Bezeichnungsschema für Objektdatentypen wird hier ob des geringen Umfangs verzichtet.

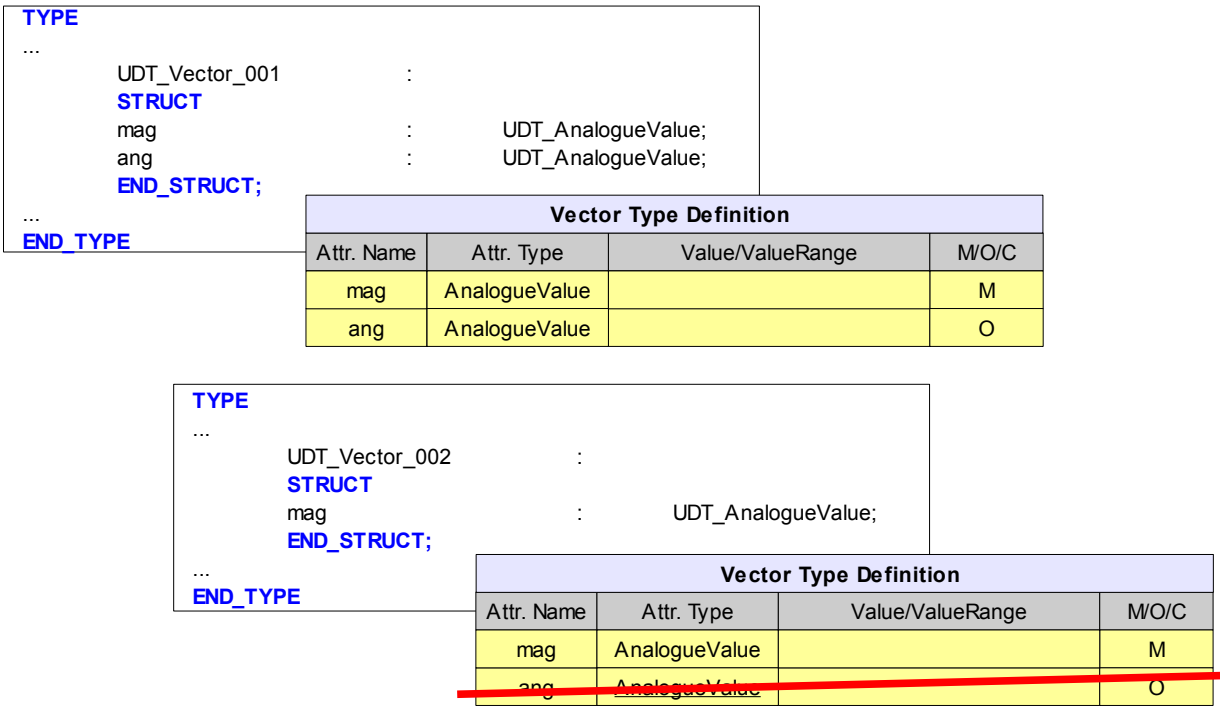

Abbildung 4.13: Abwahl von Eigenschaften eines Objektdatentyps erfordert neue Datentypen in ST

Wie jedoch schon erläutert, spielt die eindeutige Bezeichnung von Objektdatentypen bei Objekten unterhalb der Logical Nodes eine Rolle, da sie über gleich lautende Namen angesprochen werden. So ist der in einer Struktur definierte Objektdatentyp Vector in der Variante *UDT_Vector_001* später über ein namens gleiches Attribut *instCVAL* ansprechbar. Die Benennung mit *instCVAL* ist festgelegt, könnte unter anderen Bedingungen aber auch eine Datenstruktur UDT_Vector_002 bezeichnen. Die Definition der nächsten Ebene mit dem Objektdatentyp CMV berücksichtigt diesen Fall mit dem vorgenannten Attribut instCVAL (Text 4.10).

Nach der Komplettierung des Datenmodells ergibt sich eine Datenstruktur, wie sie im Quellcode im Anhang ab S. 110 dargestellt ist. Diese Datenstruktur enthält bereit die für das Datenaustauschmodell notwendigen Metadaten.

4.5.5.4 Zugriff auf das Datenmodell

Nach der Erstellung des Datenmodells ist eine Methode für den Datenzugriff wünschenswert. Es gilt dabei die im einkommenden Telegramm der Nachrichtenübermittlung enthaltenen Parameter zu nutzen, die für das Datenmodell relevant sind. In einem ersten Schritt handelt es sich nur um die übermittelten Parameter. Ein Weg dies zu erreichen ist die Verwendung des Process Data Directory (PDD), das in jedem RFC verfügbar ist. Beim PDD handelt es sich um eine Liste, die den Variablennamen dem Variablenwert gegenüberstellt. Variablen, die in der Entwicklungsumgebung PCWORX bei der Zuweisung des Typs entsprechend deklariert werden, sind so für die Funktionen und Funktionsbausteine RD_*_BY_SMB und WR_*_BY_SMB sichtbar. Dies gilt auch bei der Zuweisung mit STRUCT definierter Datenstrukturen, die das Datenmodell darstellen.

Die Funktionen und Funktionsbausteine RD_*_BY_SMB und WR_*_BY_SMB erlauben den lesenden und schreibenden Zugriff auf Variablen, wenn der Variablenname als String übergeben wird. Es kann sich dabei auch um ein Variablenfeld innerhalb einer verschachtelten Struktur handeln. Der Patzhalter „*“ steht für den Datentyp der angesprochenen Variablen, die es zu lesen oder zu schreiben gilt. Zusätzlich stellen diese Funktionen und Funktionsbausteine Fehlerflags zur Verfügung, die Unauffindbarkeit oder den Zugriff auf eine Variable mit unpassenden Datentyp signalisieren.

Ein Vorteil ergibt sich aus der Tatsache, dass die Notation der Parameter des vereinbarten Telegrammformates direkt für den Zugriff auf Datenfelder in den ST Strukturen genutzt werden kann. Dazu muss die Objektreferenz des betreffenden Parameterstrings nach der Extraktion aus dem Telegramm um eine Programm interne Referenz ergänzt werden. Diese Programm interne Referenz steht zur Verfügung, nachdem die das Datenmodell repräsentierende Datenstruktur einer Variable zugewiesen wurde. Der Name dieser Variablen entspricht dem Physical Device. Nach Abbildung 4.12 trägt die Variable den Namen *RFC01*. Da sämtliche Funktionalitäten im Baustein IEC61400_25_V1_00 gekapselt sind, wird diese Variable dort angelegt und zusätzlich dem PDD bekannt gemacht. Es ist nach IEC 61131 nicht möglich, Funktionsbausteine allein auszuführen. Sie sind zwingend aus einem Hauptprogramm heraus auszuführen, wobei das Programm einer SPS-Task zugewiesen sein muss. In dieser Teilimplementierung wird deshalb ein Hauptprogramm *Main* der Default-Task zugewiesen. In *Main* wird eine Instanz *iec61400_25* von IEC61400_25_V1_00 angelegt und in jedem SPS-Zyklus ausgeführt. Die für die Nutzung der Funktionen und Funktionsbausteine RD_*_BY_SMB und WR_*_BY_SMB ergeben sich nun aus dem Namen des Hauptprogramms *Main* und der Funktionsbausteininstanz *iec61400_25*. Diese Ergänzungen werden

jedoch Programm intern vorgenommen und müssen nicht Teil des Telegramms sein. Ein Beispiel für die Bildung einer Programm internen Objektreferenz für den Zugriff auf eine Variable aus dem Datenmodell zeigt Abbildung 4.14.

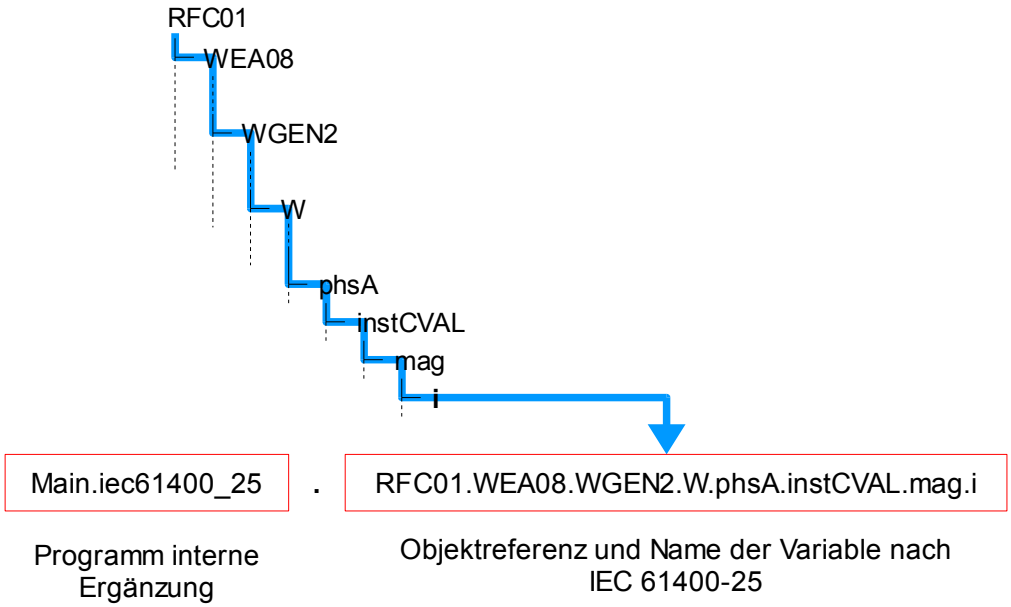

Abbildung 4.14: Bildung der Programm internen Objektreferenz

4.5.6 Umsetzung des Datenaustauschmodells

4.5.6.1 Organisation der ACSI Services

Nach der Umsetzung des Datenmodells erfolgt die Implementation der ACSI-Services. Für die Implementation wurden alle Services der Klassen Server, Logical Device, Logical Node und Data berücksichtigt (siehe [IEC400-25-3], S. 20/21). Es handelt sich um die Services:

- GetServerDirectory,
- GetLDDirectory,
- GetLNDirectory,
- GetDataDefinition,
- GetDataDirectory,
- SetDataValues sowie
- GetDataValues.

Bei der Auswahl wurden Services gewählt, die zur Arbeit mit dem Datenmodell mindestens notwendig sind. Services, die auf Datasets wirken ließen sich auf gleiche Weise implementieren.

Services sind an Objekte gebunden, wie Eigenschaften im Datenmodell auch. Sie wirken nur mit einem vorhandenen Datenmodell. Der Weg dorthin unterscheidet sich jedoch von dem zu Datenmodell aufgrund der bereits dokumentierten Einschränkungen bei der Kapselung von Fähigkeiten. Die Services können deshalb nicht direkt mit den Funktionen und Funktionsbausteinen RD_*_BY_SMB und WR_*_BY_SMB angesprochen werden, wie es beim Datenmodell möglich ist. Die Lösung kann darin gefunden werden, den Nutzdaten des Datenmodells Metadaten hinzuzufügen.

Neben der Notwendigkeit, Fähigkeiten den Klassen zuzuordnen, ist auch deren mögliche Abwahl zu berücksichtigen, so sie nicht mit dem Attribut „Mandatory“ gekennzeichnet sind. Dabei ist zu beachten, dass die Services auf das durch die Objektreferenz gekennzeichnete Objekt angewandt werden. Für die Umsetzung dieser Punkte stehen der Servicename und dessen Objektreferenz nach der Zerlegung des Nachrichtentelegramms in seine Bestandteile zur Verfügung.

4.5.6.2 Nutzung System eigener Dienste des RFC

Die IEC 61400-25 erlaubt die Nutzung System eigener Dienste bei der Umsetzung der zu implementierenden Services. Die Tabelle 4.4 beinhaltet eine Gegenüberstellung der Datenaustauschmodelle und der ACSI-Servicemodelle nach [IEC400-25-3], Table 1. Zusätzlich wurde die letzte Spalte um die Fähigkeit des RFC ergänzt, die entsprechenden ACSI-Servicemodelle durch eigene Funktionen zu gewährleisten oder zu unterstützen. In der Gruppe der Managementfunktionen existieren drei Datenaustauschmodelle, die ausdrücklich System spezifisch ausgeführt sein können. Für die übrigen Datenaustauschmodelle sind entsprechende Implementierungen nach [IEC400-25-3] und [IEC850-7-2] vorzusehen. Dabei kommt Strukturierter Text (*ST*) aufgrund seiner Flexibilität grundsätzlich als bevorzugte Variante für die Implementierung in Frage. Darüber hinaus können Firmwaredienste (*FD*), die durch das Betriebssystem des RFC angeboten werden, in eine Implementierung durch ST eingebunden werden. Dadurch würde die Komplexität und der Aufwand der Implementierung selbst vermindert. Zu guter Letzt erlaubt [IEC400-25-3] bzw. [IEC400-25-4] (S. 236) eine Zeitsynchronisation des Physical Devices mit einem externen SNTP-Server.

Für die Umsetzung der in Abschnitt 4.5.6.1 benannten Services ergibt sich daher keine Nutzung System interner Dienste.

Funktions-gruppe	Daten-austausch-model	Kurz-beschreibung	Informations-kategorien	Übertragungs-prinzip	ACSI-Service-model	RFC
Operational	Autorisation	Zugangs-management	Kurze Textnachrichten	Kommando-übertragung bei Bedarf Sollwert-übertragung	ASSOCIATION	ST
	Steuerung	Steuerung Geräte	Sollwerte Kommandos	Kommando-übertragung Sollwert-übertragung	CONTROL	ST
	Beobachtung	Beobachtung derzeitiger Daten und Datenänderung der Geräte	Messdaten Prozessdaten Status Alarm Ereignisse Zeitgeber Zähler Sollwerte Parameter Kommandos Zeitreihen	Periodische Daten-übertragung Daten-übertragung bei Bedarf Ereignis-gesteuerte Daten-übertragung	LOGICAL-DEVICE LOGICAL-NODE DATA DATA-SET BUFFERED-REPORT-CONTROL UNBUFFERED-REPORT-CONTROL LOG LOG-CONTROL	ST
	Reporting und Logging	Triggergesteuerte und kontinuierliche Aufzeichnung von Daten und Ereignissen	Logs Reports Statistiken Kurven Trends Ereignisse Kurze Textnach-richten			ST
Functional	Diagnose	Selbstbeobach-tung der Geräte	Beobachtung sowie Reporting und Logging anwenden			FD
	Benutzer- und Zugriffs-verwaltung	Einstellungen der Benutzer, Zugriffsberechtig ungen und Beobachtungs-rechte	System spezifisch			ST
	Einstellungen	Geräte-einstellungen	System spezifisch			FD
	Zeit-synchronisation	Synchronisation der Uhren	SCSM spezifisch			SNTP

Tabelle 4.4: Datenaustauschmodelle und ACSI-Servicemodelle nach [IEC400-25-3] nebst Fähigkeiten RFC

4.5.6.3 Implementierung von Services zur Datenmanipulation

Services, die in der Data Class und deren Ableitungen ([IEC850-7-2], Figure 13) implementiert werden, dienen zur Datenmanipulation und zum Datenmanagement (Tabelle 4.5). Common Data Classes sind Ableitungen der Data Class. Es ist möglich, einige Services abzuwählen, nicht als „Mandatory“ gekennzeichnet sind. In der Data Class trifft dies auf *GetDataDirectory* und *GetDataDefinition* zu.

Service	*Beschreibung*
GetDataDirectory	Auslesen der Daten
SetDataValues	Schreiben der Daten
GetDataValues	Auslesen der Objektreferenzen
GetDataDefinition	Beschreibungen der Datentypen

Tabelle 4.5: Services der Data Class

Um diese Services jedoch an eine Data Class zu binden, sind zusätzliche Metadaten anzulegen. Dies bedeutet, zusätzliche Felder in das Datenmodell einzubringen. Anhand der Implementierung von *GetDataValues* wird dies demonstriert.

Für die Zugehörigkeit des Services zum Datenobjekt wird eine zusätzlichen Variable des Datentyps BOOL angelegt. Die Variable trägt den Namen des Services *GetDataValues*. Ist der Service als „Mandatory“ gekennzeichnet oder angewählt, wird der Wert TRUE zugewiesen. Das Vorgehen ermöglicht die erneute Anwendung des PDD. Zur Ermittlung eines boolschen Datentyps steht ein Funktionsbaustein RD_BOOL_BY_SMB zur Verfügung. Durch Ermittlung der Objektreferenz der Services und des Servicenamens aus dem Telegramm, kann diese Zeichenkette direkt zur Abfrage des Inhalts der Variable gleichen Namens genutzt werden. Jedoch ist auch hier die Programm interne Objektreferenz zu ergänzen.

Parameter name
Request
Reference
Request+
DataAttributeValue [1..n]
Request-
ServiceError

Tabelle 4.6: Parametertabelle für GetDataValues nach [IEC850-7-2], S. 51

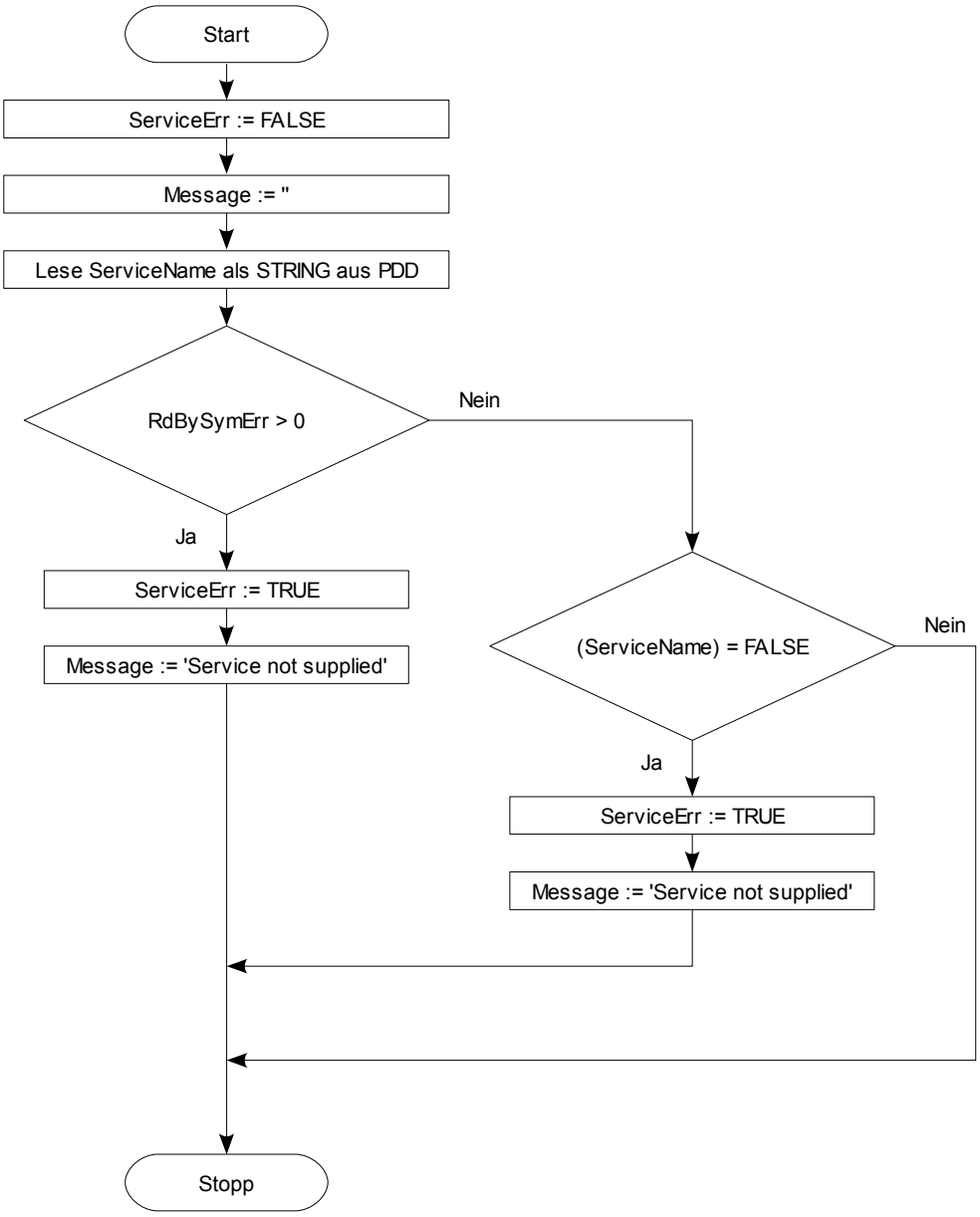

Abbildung 4.15: Algorithmus zur Ermittlung der Verfügbarkeit eines Services

Durch Abfrage des Fehlerflags (*RdBySymErr*) ist die Information über die Verfügbarkeit in dieser Klasse erhältlich, wenn der Service fehlerhafterweise auf ein Datenobjekt angewandt wird, welches nicht vom Objektdatentyp Data ist. Ist die Variable vorhanden, weist aber den Wert FALSE auf, so ist der Service bei der Codeerstellung abgewählt worden. Die im Fehlerfall (*Response-*) zu sendende Parameter sind in der für diesen Service gültigen Parametertabelle (Tabelle 4.6) zu suchen. Der *ServiceError* ist nicht fix und kann auch in Form einer Klartextmitteilung erstellt werden (Abbildung 4.15). Für die Programm interne Verwendung wird zusätzlich ein Fehlerflag ServiceErr gesetzt.

Für *Response+*, der Service Antwort bei erfolgreicher Ausführung, ist die Rücklieferung der

Variablenwerte vorgesehen. In dieser Teilimplementierung ist vorgesehen, auch komplexe Objektdatentypen einzeln und nicht in Gruppen abzufragen (Abbildung 4.16). Bestehen die Eigenschaften nicht aus Basisdatentypen, sondern aus komplexen Datentypen, so sind die darin enthaltenen Felder mit Basisdatentypen durch die Objektreferenz anzusprechen. Es wird die Kenntnis der Struktur der komplexen Datentypen vorausgesetzt, da diese Datentypen selbst nicht als Klassen definiert sind und keinerlei Services aufweisen.

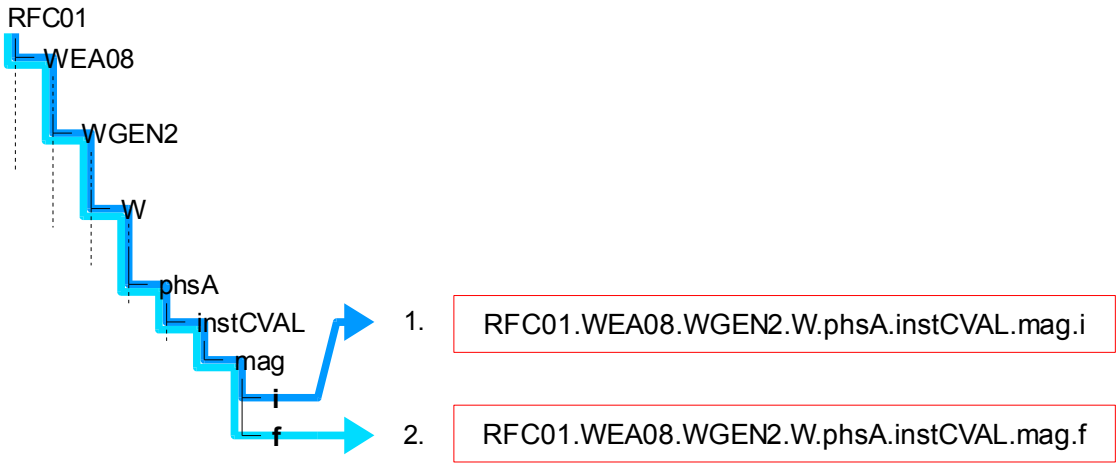

Abbildung 4.16: Datenzugriff auf ...mag erfordert zwei Schritte

Für den Datenzugriff wird in der Teilimplementierung ferner angenommen, dass nur die ST Datentypen INT, REAL und STRING(255) verwendet werden, was den IEC Basisdatentypen INT32, FLOAT und VISIBLESTRING255 entspricht. Der Zugriff erfolgt dabei wieder mit den Funktionen RD_*_BY_SMB unter Verwendung der mit der Funktion übergebenen Parameter. Es ist dazu nicht notwendig, den Datentyp im Einzelfall zu kennen. Die Anwendung einer für den Datentyp nicht passenden Funktion RD_*_BY_SMB ist über das Fehlerflag *RdBySymErr* abzufragen. Dadurch ergibt ich die Möglichkeit, alle bekannten Funktionen zu probieren, bis das Fehlerflag nicht mehr gesetzt wird (Abbildung 4.17).

Neben dem Service *GetDataValues* zur Abfrage der Daten aus dem Datenmodell existiert ein entsprechender Service *SetDataValues* zum Setzen und Ändern von Variablenwerten, der auch als „Mandatory“ gekennzeichnet ist. Der Algorithmus unterscheidet sich in nur zwei Punkten von dem des Services *GetDataValues*.

Die erste Abweichung zum Algorithmus von *GetDataValues* entsteht folglich aus der entgegen gesetzten Aufgabe, Daten zu setzen. Daher sind die Funktionen RD_*_BY_SMB durch WR_*_BY_SMB zu ersetzen. Auch diese Funktionen sind über ein Fehlerflag abzufragen und verhalten diesbezüglich äquivalent. Das Fehlerflag heißt hier jedoch WrBySymErr.

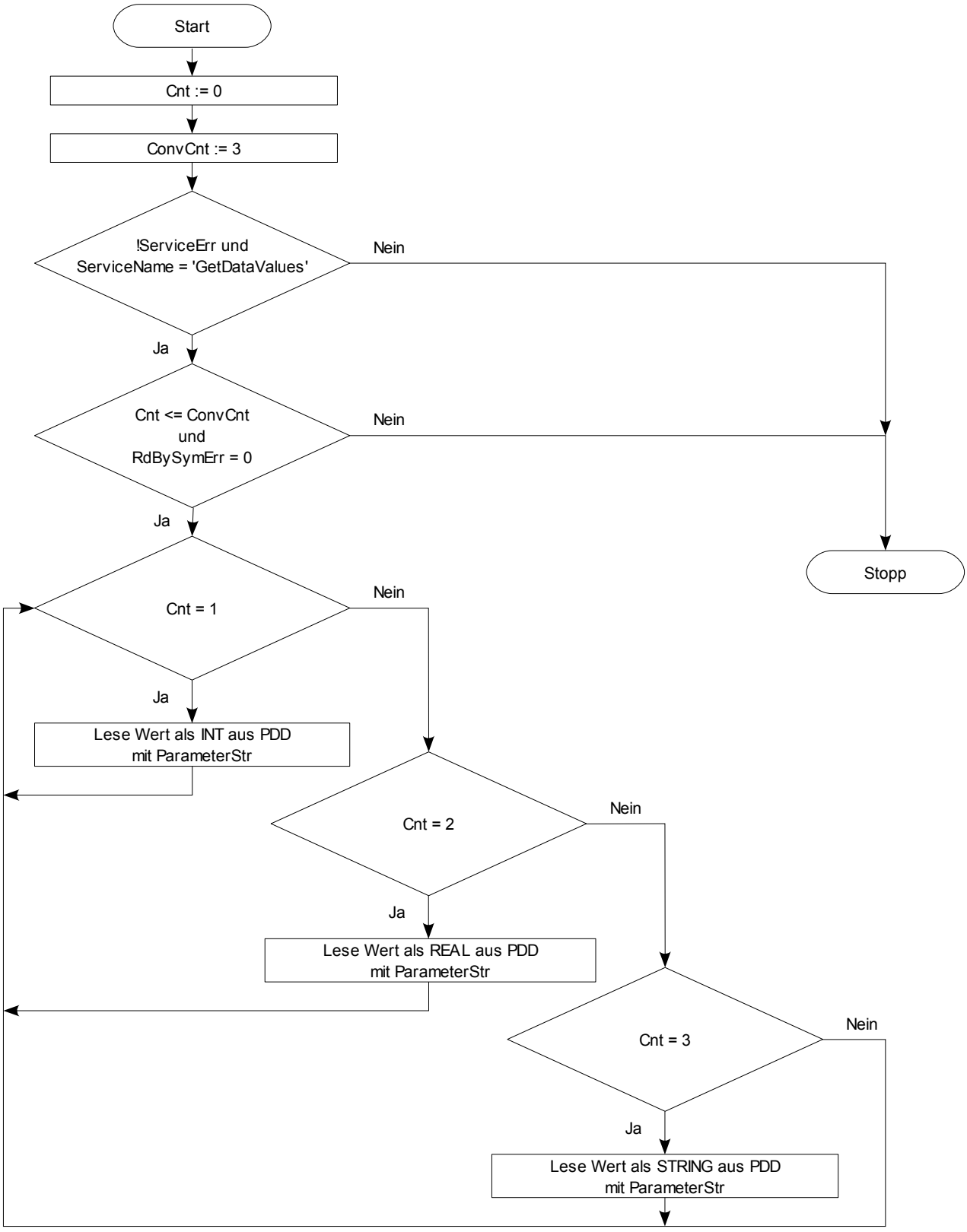

Abbildung 4.17: Algorithmus des Services GetDataValues

Der zweite Unterschied zum vorgenannten Algorithmus liegt in der Behandlung der Übergabeparameter und Rückgabewerte. Die Parametertabelle *SetDataValues* für den Service ist Tabelle 4.7 dargestellt. Während die Behandlung eines Fehlers wird identisch gehandhabt wird, existiert im Erfolgsfall kein Rückgabewert. Es bietet sich daher an, den geänderten Wert nach Ausführung des Services durch eine *GetDataService* zu kontrollieren, um vom Service nicht erkannte Fehler zu detektieren.

Parameter name
Request
Reference
DataAttributeValue [1..n]
Request+
Request-
ServiceError

Tabelle 4.7: Parametertabelle für SetDataValues nach [IEC850-7-2], S. 52

Die Übergabeparameter werden, wie beim Service *GetDataValues*, auf ein Feld des Datenmodells beschränkt. Damit ist die Implementierung im Rahmen dieser Arbeit abgeschlossen. Es verbleibt die Umsetzung der Services, die Informationen über das Datenmodell liefern.

4.5.6.4 Implementierung von Services zur Informationsakquise des Datenmodells

Die Anwendung der Services zur Datenmanipulation erfordert die Kenntnis des Datenmodells und seiner Struktur. Für die Informationsbeschaffung existierende Services der hier verwendeten Objektdatentypen sind in Tabelle 4.8 aufgelistet. Aufgrund der sehr ähnlichen Parametertabellen, ist die Implementation in einem Schritt möglich. Tabelle 4.9 zeigt die Parametertabelle der Services *GetServerDirectory* beispielhaft. Grundsätzlich wird ein Datenobjekt übergeben, dessen

Service	*Beschreibung*	*Class*
GetDataDirectory	Listet alle Data Attribute	Data
GetDataDefinition	Beschreibungen der Datentypen	Data
GetLNDirectory	Listet alle Data Classes	Logical-Node
GetLDDirectory	Listet alle Logical Nodes	Logical-Device
GetServerDirectory	Listet alle Logical Devices	Server

Tabelle 4.8: Services für die Informationsbeschaffung des Datenmodells

untergeordnete Datenobjekte aufzulisten sind. Im Fehlerfall ist eine adäquate Fehlermeldung zu generieren.

Für die Implementierung kann erneut auf die Funktionen RD_*_BY_SMB und WR_*_BY_SMB zurück gegriffen werden. Allerdings existieren keine Funktionen oder Funktionsbausteine, die Informationen über Struktur oder Datentypen einer Struktur liefern. Diese Problem ist zu lösen, in dem weitere Metadaten in das Datenmodell eingefügt werden. Die durch den Services bei Abfrage zu übermittelnden Daten werden in zusätzlichen Variablen gespeichert. Diese Variablen werden hier als Metavariablen benannt.

Parameter name
Request
Objectclass
Request+
Reference [0..n]
Request-
ServiceError

Tabelle 4.9: Parametertabelle des Services GetServerDirectory

Die Zuordnung erfolgt auf ähnliche Weise, wie die bereits implementierte Bindung der Services zu ihren Datenobjekten. Der Unterschied liegt lediglich in der Benennung dieser zusätzlichen Metavariablen und der Art der Nutzung. Der Inhalt dieser Variablen wird direkt genutzt, d.h. die Ausführung eines Services liest den Variableninhalt aus und sendet diesen als Rückgabewert an den Client. Unter Berücksichtigung der o.g. Funktionen kann auch die Zugehörigekeit dieser Variablen zu jeweiligen Service einfach dadurch geregelt werden, dass die Benennung der Metavariablen mit der Benennung des Services abzüglich der ersten drei Zeichen („GET“) identisch ist.

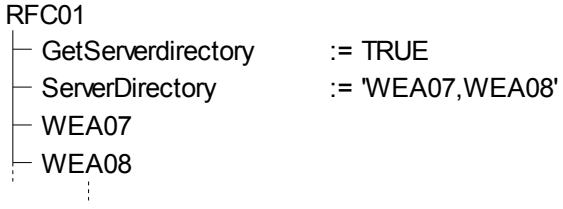

Abbildung 4.18: Ausschnitt aus einem Datenmodell

Diese sehr einfache Lösung hat den Vorteil der Lokalität und Systematik. Die Lokalitätist darin zu sehen, dass die Metadaten am Ort Ihrer Entstehung gespeichert werden. Diese Tatsache erleichtert die Beseitigung von Fehlern, da diese auch in umfangreichen Datenstrukturen einfach zu finden sind. Die Systematik kann sehr schnell darin gefunden werden, dass mit jedem Service seine

Metadaten am Ort der Anwendung im Datenmodell hinterlegt und somit auch die gewollte Objektzugehörigkeit des Services gesichert ist.

Als Beispiel Abbildung 4.18 wird der Service *GetServerDirectory* aus Tabelle 4.9 betrachtet. Wie schon erläutert, ist das Anlegen einer Variablen gleichen Namens vom Typ BOOL notwendig, um die Aktivierung des Services anzuzeigen. Der Service *GetServerDirectory* hat die Aufgabe, alle Logical-Device Klassen aufzulisten, die eine Server Klasse untergeordnet sind. Um den Mangel einer Abfrage innerhalb der Datenstruktur zu umgehen, wird das korrekte Ergebnis in der Metavariable ServerDirectory gespeichert. Die Variable ist vom Typ STRING und enthält bei mehr als einem Eintrag eine durch Komma getrennte Liste. Der Programmablaufplan zu diesem Algorithmus ist in Abbildung 4.19 dokumentiert.

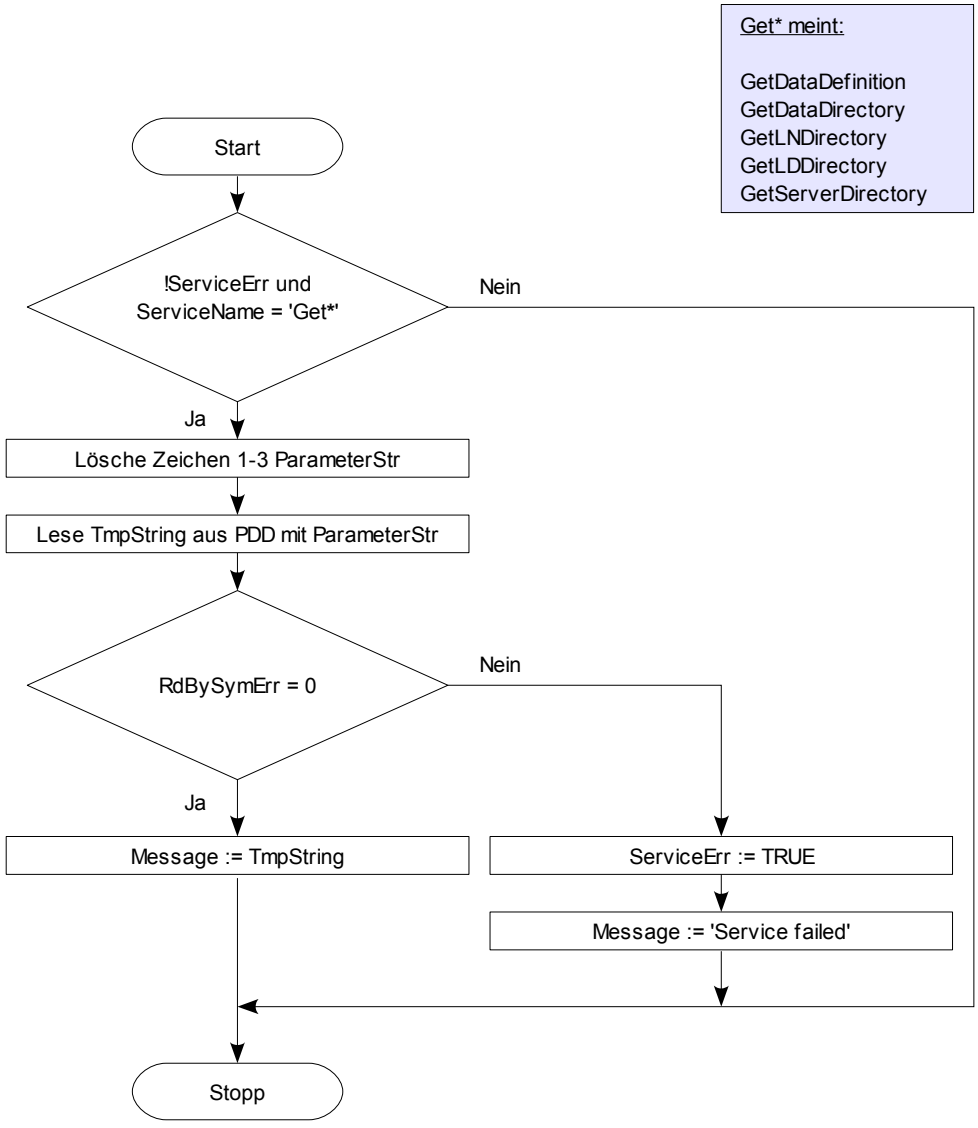

Abbildung 4.19: Algorithmus der Services GetDataDefinition, GetDataDirectory, GetLNDirectory, GetLDDirectory und GetServerDirectory

4.6 Test und Validierung

Nach der Umsetzung des Daten- und Datenaustauschmodells steht ein kleines arbeitsfähiges System zur Verfügung. Auf Basis der Vereinbarung des vereinfachten Telegrammformats ist eine Kommunikation mit Hilfe des TCP-Protokolls möglich. Dazu kan ein einfacher Client verwendet werden, der ein Telegramm übertragen und Rückgabewerte entgegen nehmen kann. Für die Test stand ein TCP-Client SimpleNet zur Verfügung, den der Autor dieser Arbeit erstellt hat.

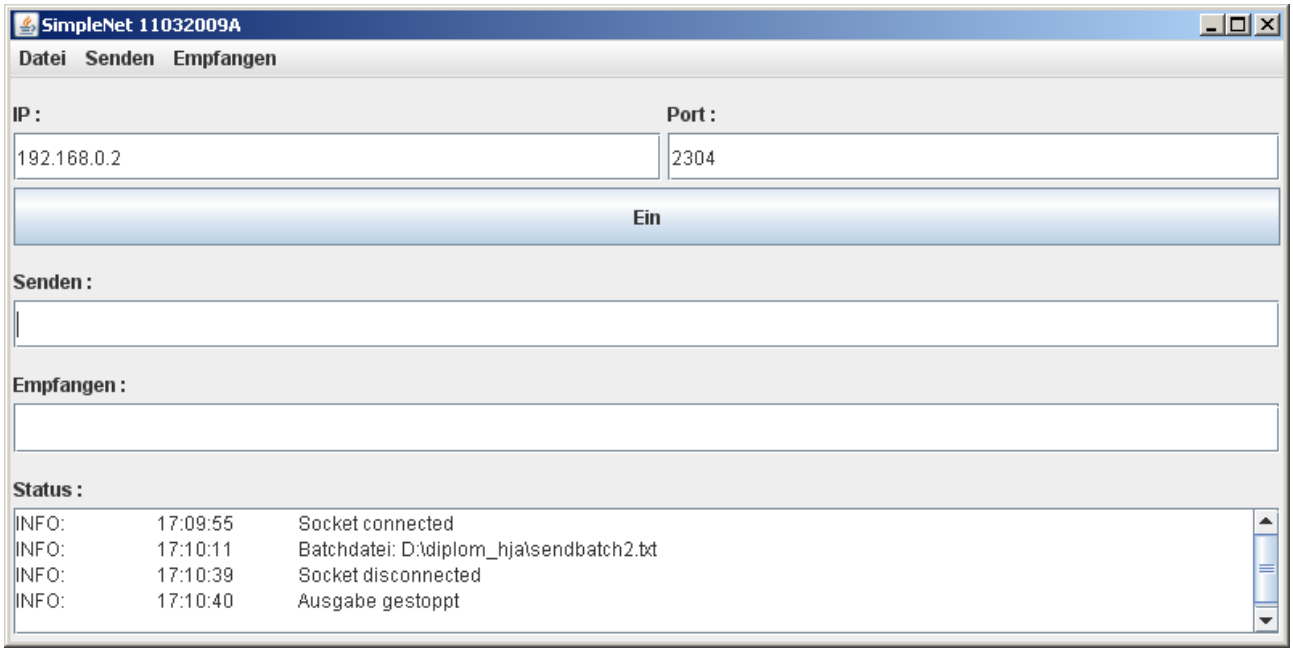

Abbildung 4.20: Testclient SimpleNet

Nach Start der SPS ist der TCP-Server bereit, Verbindungen auf der Schnittstelle 192.168.0.2 auf Port 2304 von einem TCP-Client anzunehmen. Durch Betätigung des EIN-Buttons wird die Verbindung durch den Client aufgenommen. Die Eingabe von Telegrammen ist es jetzt möglich, ein Datenmodell, wie in Abbildung 4.16 dargestellt, zu befragen und zu manipulieren. Dazu werden hier einige Beispiele demonstriert.

Die Frage nach den dem Physical Device (Server) untergordneten Logical Devices wird mit

```
RFC01.GetServerDirectory(RFC01)
```

erfragt. Die Ausgabe listet das einzige Logical Device

```
WEA08
```

auf. Wie bereits dargelegt, stammt diese Information aus einer Metavariable. Alle weiteren Services zur Informationsbeschaffung über das Datenmodell funktionieren äquivalent. Neben diesen existieren jedoch auch zwei Services zur Datenmanipulation. Durch Eingabe von

```
RFC01.WEA08.WGEN2.W.phsA.GetDataValues(RFC01.WEA08.WGEN2.W.phsA.in
stCVAL.ang.f)
```

erhält man z.B. einen Wert

```
99.01
```

da die Variable vom Typ REAL (IEC 61131) bzw. FLOAT (IEC 61400-25) ist. Für die Programm interne Ermittlung der richtigen Typkonvertierung erfolgt unter Auswertung von Fehlerflags der erläuterten RD_*_BY_SMB Funktionen. Die Ausführung von

```
RFC01.WEA08.WGEN2.W.phsA.SetDataValues(RFC01.WEA08.WGEN2.W.phsA.in
stCVAL.ang.f,25.001)
```

liefert per Definition der Serviceparametertabellen keine Rückmeldung. Die erfolgreiche Aktion kann jedoch mit einem erneuter Ausführung des Services *GetDataValues* geprüft werden.

4.7 Ausblick

Die Teilimplementierung hat sich mit den grundlegenden Mechanismen der IEC 61400-25 befasst. Darauf aufbauend gibt noch eine Reihe von Kandidaten für die Fortführung der Arbeit. Neben der Umsetzung des, allerdings relativ anspruchslosen, Benutzermanagements dieser IEC, sind Funktionen rund um Datasets, Reporten und Logs einzubetten. Neben diesen Tätigkeiten gibt es jedoch auch solche, die der Bewältigung der Modellerstellung dienen.

Bei der Erstellung von kleinen Teilmodellen wurde bereits festgestellt, dass die Zusammenstellung der Datenobjekte sehr umfangreich sein kann. Aufgrund der Komplexität der Aufgabe ist es vorausschauend, über die Umsetzung eines Codegenerators nachzudenken. Auf Basis bereits definierter Datentypen in ST kann die Zusammenstellung des Daten- und Datenaustauschmodells rascher und fehlerfreier gelingen. In diesem Zusammenhang ist durchaus denkbar, ein Tool mit Assitentencharakter zu erschaffen, da die manuelle Suche geeigneter Datenobjekte zeitlich sehr aufwendig ist und Fragen zur Aktivierung bestimmter Eigenschaften und Fähigkeiten immer wieder auftauchen. Die Existenz eines solchen Tools eröffnet zusätzlich die Möglichkeit der Integration in die Entwicklungsumgebung PCWORX.

Die Arbeit mit diesem Standard steht jedoch nicht allein. Sind diese Schritte erst getan, wird sich nach Prüfung der Praxistauglichkeit herausstellen. Hinzu kommen weitere Herausforderungen, die durch nahende Revisionen der IEC 61400-25 ankündigen.

5 Zusammenfassung

Die Teilimplementierung hat gezeigt, das die Umsetzung der IEC 61400-25 mit Phoenix Automationssystemen grundsätzlich möglich ist. Im Zuge dieser Arbeit wurden verschiedene Lösungswege für die partielle Umsetzung des Daten- und Datenaustauschmodells versucht, die sich am jeweiligen Kenntnisstand orientierten. Da sich objektorientierte Datenstrukturen nicht direkt mit die IEC 61131 abbilden lassen, ging es um Ansätze mit äußerlich gleichen Eigenschaften und Fähigkeiten.

Zum ersten Hauptergebnis dieser Arbeit zählt, dass die Umsetzung des Datenmodells mit den STRUCT Syntaxelementen zweifelsfrei machbar ist. Neben der Nachbildung der geforderten Datenstruktur, waren die Existenz äquivalenter Datentypen in ST Grundlage für die Implementierung. Darauf aufbauend sind Lösungen für die Einbindung essentieller Services unter Berücksichtigung ihrer Objektzugehörigkeit gefunden worden, indem das Datenmodell um Metadaten ergänzt wurde, die für die Funktion und Zugehörigkeit der Services notwendig sind. Jedoch muss gesagt werden, dass die ausgeprägte Spezialisierung die Formulierung einheitlicher Modellierungsregeln erschwert. Ein Weg kann darin gesehen werden, ähnliche Services bzw. Funktionalitäten ohne Rücksicht auf ihre eigentliche Objektzugehörigkeit zu gruppieren, um eine arbeitsintensive Aufreihung von Ausnahmen zu vermeiden.

Ein grundlegendes Ziel dieser Arbeit war, erste Erfahrungen mit der IEC 61400-25 zu sammeln, um eine zuverlässige Anschätzung über den zu erwartenden Aufwand einer ganzheitlichem Implementation mit Phoenix Automationsprodukten zu erhalten. Abschließend kann beurteilt werden, dass der erhebliche Umfang sowie die Art der Strukturierung verbunden mit nicht eindeutigen Formulierungen diese Aufgabe komplex und fehleranfällig werden lassen. Daraus ergab sich bei der Bearbeitung dieser Aufgabe wesentlich höherer Zeitbedarf, als zu Beginn eingeschätzt. Dem entgegen zu setzen sind vorwiegend einfache Regeln zur Implementierung, die auch bei mittleren und großen Systemen die Möglichkeit zur Kontrolle lassen. Da derzeit auch mehr Clients als Server auf dem Markt sind, bietet sich bei ernsthafter Verfolgung eines solchen Vorhabens an, den Dialog mit den Produktherstellern in diesem Zusammenhang zu suchen.

Literaturverzeichnis

[IEC400-25-1] IEC, Beuth, Communications of wind power plants - Overall description of principles and models, 2006, IEC 61400-25-1

[IEC400-25-2] IEC, Beuth, Communications of wind power plants - Information models, 2006, IEC 61400-25-2

[IEC400-25-3] IEC, Beuth, Communications of wind power plants - Information exchange models, 2006, IEC 61400-25-3

[IEC400-25-3] IEC, Beuth, Communications of wind power plants - Information exchange models, 2006, IEC 61400-25-3

[IEC400-25-4] IEC, Beuth, Communications for monitoring and control of wind power plants - Mapping to communication profile, 2008, IEC 61400-25-4, ISBN 2-8318-9964-8

[IEC400-25-5] IEC, Beuth, Communications for monitoring and control of wind power plants - Conformance testing, 2006, IEC 61400-25-5

[IEC850-7-1] IEC, DIN EN, Beuth, Kommunikationsnetze und -systeme in Stationen - Grundlegende Kommunikationsstruktur für stations- und feldbezogene Ausrüstung - Grundsätze und Modelle, 2004, DIN EN 61850-7-1

[IEC850-7-2] IEC, DIN EN, Beuth, Kommunikationsnetze und -systeme in Stationen - Grundlegende Kommunikationsstruktur für stations- und feldbezogene Ausrüstung - Abstrakte Schnittstelle für Kommunikationsdienste (ACSI), 2004, DIN EN 61850-7-2

[IEC850-7-3] IEC, DIN EN, Beuth, Kommunikationsnetze und -systeme in Stationen - Grundlegende Kommunikationsstruktur für stations- und feldbezogene Ausrüstung - Gemeinsame Datenklassen, 2004, DIN EN 61850-7-3

[IEC850-7-4] IEC, DIN EN, Beuth, Kommunikationsnetze und -systeme in Stationen - Grundlegende Kommunikationsstruktur für stations- und feldbezogene Ausrüstung - Kompatible Logikknoten- und Datenklassen, 2004, DIN EN 61850-7-4

[IEC850-8-1] IEC, DIN EN, Beuth, Kommunikationsnetze und -systeme in Stationen - Spezifische

Abbildung von Kommunikationsdiensten (SCSM) - Abbildung auf MMS (nach ISO 9506-1 und ISO 9506-2) und ISO/IEC 8802-3 (IEC61850-8-1:2004), 2004, DIN EN 61850-8-1

[GOE99] Göbel, Jürgen; 969 S, Hüthig, Kommunikationstechnik, 1999, ISBN 3-7785-3904-3

[HEI03] Heier, Siegfried ; 555 S, Teubner , Windkraftanlagen, 2003, ISBN 3-519-26171-5

[GAS07] Gasch, Robert; Twele, Jochen; 569 S, Teubner, Windkraftanlagen, 2007, ISBN 978-3-8351-0136

[BRA07] Brause, Rüdiger; 262 S, Springer, Kompendium der Informationstechnologie, 2007, ISBN 978-3-540-27061-4

[IECWWW] www.iec.ch, IEC, 2008

[PRI98] Prinz, Peter; Kirch-Prinz, Ulla; 668 S, Prentice Hall, Objektorientiert programmieren mit ANSI C+, 1998, ISBN 3-8272-9560-2

[PHOE08] Phoenix Contact GmbH & Co. KG, PC WORX IEC 61131 Programmierung, 2008

[SEGU06] Seemann, Jochen;von Gudenberg, Jürgen Wolff; 359 S, Springer, Software-Entwurf mit UML 2, 2006, ISBN-10 3-540-30949-7

[BRPK08] Brandt-Pook, Hans; Kollmeier, Rainer; 190 S, Vieweg+Teubner, Softwareentwicklung kompakt und verständlich, 2008, ISBN 978-3-8348-0365-8

[SCHO05] Scholz, Peter; 232 S, Springer, Softwareentwicklung eingebetteter Systeme, 2005, ISBN3-540-23405-5

[WUES06] Wüst, Klaus; 294 S; Vieweg, Mikroprozessortechnik, 2006, ISBN 978-3-8348-0046-6

[BEHA04] Beierlein, Thomas; Hagenbruch, Olaf; 600 S, Fachbuchverlag Leipzig, Taschenbuch Mikroprozessortechnik, 2004, ISBN 3-446-22072-0

[RFC09] Phoenix Contact GmbH & Co. KG, Installation und Betrieb des Remote Field Controllers RFC 470 PN 3TX, 2009

[SCWI06] Schnell, Gerhard (Hrsg.); Wiedemann, Bernhard (Hrsg.); 414 S, Vieweg, Bussysteme in derAutomatisierungsundProzesstechnik, 2006, ISBN 3-8348-0045-7

[LOV07] Love, Jonathan; 1093 S, Springer, Process Automation Handbook, 2007, ISBN 9781846282812

[HPVHB05] Holleczek, Peter; Vogel-Heuser, Birgit (Hrsg); 146 S, Springer, Echtzeitaspekte bei der Koordinierung Autonomer Systeme, 2005, ISBN 10 3-540-29594-1

Anhang

Quellcode des Funktionsbausteins IEC61400_25_V1_00

```
(*@PROPERTIES_EX@

TYPE: POU

LOCALE: 0

IEC_LANGUAGE: ST

PLC_TYPE: independent

PROC_TYPE: independent

*)

(*@KEY@:DESCRIPTION*)

************************************************************************
```

****

Name : IEC61400_25_Stat_V1_00

State : [ ] demo [ ] released

Develop Environment :

PC WORX : PC WORX 5.2

Controller : unbestimmt

Devices : RFC

****

Change Notes:

Date	Version	Author	Description
17.02.09	1.00	PxCE/HJA	Initial version

```
(*@KEY@:END_DESCRIPTION*)
FUNCTION_BLOCK IEC61400_25_V1_00

(*Group:Input Parameters*)


(*Group:Output Parameters*)
```

(*Group:InOut Parameters*)

(*Group:Local Variables*)

```
VAR
    StartDataSending :    BOOL := false;(*Flag f□r Abschluss der Datenverarbeitung*)
    TcpOutStruc : UDT_TcpOut;(*Zeichenkette, die dem IEC61400-25 Client auf seine Anfrage gesandt wird*)
    IpSndErr :    BOOL;(*Flag wird gesetzt, wenn ein Problem beim TCP-Versand auftrat*)
    IpSndStat :   INT;(*Status der TCP Versandseinheit*)
    IpConValid :  BOOL := false;(*Flag f□r eine IP Connection*)
    IpConStatus : INT;(*Status der IP Connection*)
    IpConId :     INT;(*ID der IP Connection*)
    IpConErr :    BOOL;(*Flag zur Erkennung eines Fehlers der IP connection*)
    IpRsvStat :   INT;(*Status der TCP Empfangseinheit*)
    IpRsvDone :   BOOL;(*Flag wird gesetzt, wenn das TCP Telegramm empfangen wurde*)
```

IpRsvErr : BOOL;(*Flag wird gesetzt, wenn ein Problem beim TCP-Empfang auftrat*)

TcpInStruc : UDT_TcpIn;(*Buffer enth□lt die empafngenen TCP-Zeichen und die Telegramml□nge*)

TcpIn : UDT_str_255;(*Sammelt die empfangenen Zeichen zu einem String*)

IpSndDone : BOOL;(*Flag wird gesetzt, wenn das TCP Telegramm versandt wurde*)

BufCpy : INT;(*R□ckgabewert der Funktion BufCpy*)

ServiceName : STRING;(*Name des angefragten Services*)

ServiceMsg : STRING;(*Fehlermeldung, wenn der Service aus irgend einem Grund nicht ausgef□hrt werden kann*)

ServiceErr : BOOL;(*Wird auf true gesetzt, wenn der Service nicht verf□gbar ist oder die Ausf□hrung negativ erfolglos war*)

Cnt : INT;(*Z□hler*)

IntOfWantedChar : INT;(*ASCII Wert eines gesuchten Zeichens*)

PointPosi : INT;(*Posi eines Punktes in der Nachricht*)

RdBySymErr : INT;(*Errorvariable des FB/FC RD_*_BY_SYM*)

RefOffset : UDT_str_255 := 'Main.iec61400_25.';(*Die Objektrefernz des IEC-Datenmodells muss um diesen String erg□nzt werden, um die interne Referenz f□r RD_*_By_SYM zu erhalten.*)

ServiceStr : UDT_str_255;(*Service mit kompletter Objektreferenz ohne Parameter*)

ParameterStr : UDT_str_255;(*Parameter des angefragten Services mit Objektreferenz*)

TcpInChr : STRING;

```
        RFC01 :         UDT_Server_001 {PDD} ;(*Instanz des Datenmodells*)
        init :    BOOL := false;(*Flag, dass die einmalige Init abgeschlossen ist*)
        ConvCnt :       INT;(*Anzahl der hier m□glichen Typkonvertierungen des R□ckgabewertes der hier implementierten ACSI Services*)
        TmpServiceMsg :     STRING;(*Nimmt den R□ckgabewert des ASCI Services kurzzeitig auf*)
        WrBySymErr :        INT;(*Errorvariable des FB/FC WR_*_BY_SYM*)
        ParamAttr :     STRING;(*F□r Set* Services gedacht, speichert die □ndernde Variable*)
        ParaValue :     STRING;(*F□r Set* Services gedacht, speichert den Wert f□r die zu □ndernde Variable*)
        TmpString :     STRING;(*Kurzzeitige Speicherung diverser Strings*)
END_VAR


(*Group:FB Instances*)


VAR
        IP_CONNECT_1 :      IP_CONNECT;(*Instanz f□r die IP-Connection*)
        IP_USEND_1 :        IP_USEND;(*TCP/IP-Versand*)
```

```
        IP_URCV_1 : IP_URCV;(*empfangene Zeichen der TCP-Verbindung*)
        RD_BOOL_BY_SYM_1 :    RD_BOOL_BY_SYM;
END_VAR



(*Group:Test*)



(*@KEY@: WORKSHEET
NAME: IEC61400_25_V1_00
IEC_LANGUAGE: ST
*)
(*
##########################################################################################
INITIALISIERUNG
##########################################################################################
*)
```

```
if not init then

(*''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''*)

    (* Services de- und aktivieren, Servicedaten einrichten *)

    (* CDC: CMV *)
    RFC01.WEA08.WGEN2.W.phsA.GetDataValues       :=    true;
    RFC01.WEA08.WGEN2.W.phsA.SetDataValues       :=    true;
    RFC01.WEA08.WGEN2.W.phsA.GetDataDirectory    :=    true;
    RFC01.WEA08.WGEN2.W.phsA.GetDataDefinition   :=    true;
    RFC01.WEA08.WGEN2.W.phsA.DataDirectory       :=    'instCVAL';
    RFC01.WEA08.WGEN2.W.phsA.DataDefinition      :=    'Vector';

    (* CDC: WYE *)
    RFC01.WEA08.WGEN2.W.GetDataValues            :=    true;
```

```
RFC01.WEA08.WGEN2.W.SetDataValues          :=    true;
RFC01.WEA08.WGEN2.W.GetDataDirectory       :=    true;
RFC01.WEA08.WGEN2.W.GetDataDefinition      :=    true;
RFC01.WEA08.WGEN2.W.DataDirectory          :=    'd,phsA';
RFC01.WEA08.WGEN2.W.DataDefinition         :=    'VISIBLESTRING255,CMV';


(* LN: WGEN2 *)
RFC01.WEA08.WGEN2.GetLNDirectory           :=    true;
RFC01.WEA08.WGEN2.LNDirectory                    :=    'NamPlt,Spd,W';

(* LD: WEA08 *)
RFC01.WEA08.GetLDDirectory                       :=    true;
RFC01.WEA08.LDDirectory                                :=    'LLN0,LPHD,WGEN1,WGEN2';

(* PHD: RFC01 *)
RFC01.GetServerDirectory                   :=    true;
```

```
    RFC01.ServerDirectory                       :=      'WEA08';

(*''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''*)

    (* Startwerte vorgeben *)

    init                                            :=      true;

    RFC01.WEA08.WGEN2.W.phsA.instCVAL.ang.i     :=      23;
    RFC01.WEA08.WGEN2.W.phsA.instCVAL.ang.f     :=      99.12;

    RFC01.WEA08.WGEN2.W.d                           :=      'Tervetuloa';

(*''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''*)

end_if;
```

```
(*
##########################################################################################
Aufbau einer IP Connection
##########################################################################################
*)

if not IpConValid then
        IP_CONNECT_1(
                EN_C                    :=      TRUE,
                PARTNER                 :=      '/PASSIVE /PORT=2304');
        IpConValid                      :=      IP_CONNECT_1.VALID;
        IpConErr                        :=      IP_CONNECT_1.ERROR;
        IpConStatus                     :=      IP_CONNECT_1.STATUS;
        IpConId                         :=      IP_CONNECT_1.ID;
end_if;

(*
```

```
##########################################################################################

EINGABE: TCP Zeichenempfang

##########################################################################################

*)


IP_URCV_1(
        EN_R                    :=      true,
        ID                      :=      IP_CONNECT_1.ID,
        RD_1                    :=      TcpInStruc);
IpRsvDone                       :=      IP_URCV_1.NDR;
IpRsvErr                        :=      IP_URCV_1.ERROR;
IpRsvStat                       :=      IP_URCV_1.STATUS;
TcpInStruc                      :=      IP_URCV_1.RD_1;


(*

##########################################################################################

TELEGRAMMAUSWERTUNG: Zerlegung in Service- und Parameterstring
```

```
##########################################################################################
*)


if IpRsvDone then
        (* Telegramm zusammenf□gen *)
        TcpIn                           :=      '';
        for Cnt := 0 to TcpInStruc.Len - 1 do
                TcpInChr                :=      BYTE_TO_STRING(TcpInStruc.RsvChr[Cnt],'%c');
                (* Service einkassieren *)
                if EQ_STRING(TcpInChr,'(') then
                        ServiceStr      :=      TcpIn;
                        TcpIn           :=      '';
                (* Serviceparameter einkassieren *)
                elsif EQ_STRING(TcpInChr,')') then
                        ParameterStr    :=      TcpIn;
                else
                (* Die Klammern brauchen wir sowieso nicht, schnippschnapp *)
```

```
                TcpIn             :=    CONCAT(TcpIn, TcpInChr);
            end_if;
        end_for;

(*''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''*)

        (* Objektreferenz des Datenmodells zur tats□chlichen Referenz dieser Implementation erg□nzen
        RefS.XXXXXXXXXXXXXXXXXXXXXX
        RefOffset.RefS. *)
        ServiceStr                  :=    CONCAT(RefOffset, ServiceStr);
        ParameterStr          :=    CONCAT(RefOffset, ParameterStr);

(*''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''*)

        (* Servicenamen ohne Objektreferenz ermitteln *)
        (* Ermittlung der Positionen des Punktes *)
```

```
ServiceName                 :=      ServiceStr;


for Cnt := 1 to len(ServiceName) do
        IntOfWantedChar     :=      GET_CHAR(
                                        ServiceName,
                                        Cnt);


        (* Der Punkt hat ASC 46 *)
        if IntOfWantedChar = 46 then
                PointPosi   :=      Cnt;
        end_if;
end_for;


(*
String links vom ServiceName loeschen, letzter Punkt
RefS.Service
XXXXXService
```

```
        *)

        ServiceName                     :=      delete(
                                                ServiceName,                    (* Zeichenfolge z. Bearbeitung*)
                                                PointPosi,                      (* Anz Zeichen zu loeschen *)
                                                1);                             (* 1. Zeichn z. Loeschen
*)

(*
##########################################################################################
ACSI: Pr□fung der Services und Ausf□hrung
##########################################################################################
*)

        (* Feststellung, ob ein Service in dem referenzierten Objekt verf□gbar ist *)

        (* Voreinstellung *)
```

```
ServiceErr              :=      false;
ServiceMsg              :=      '';


(* Geh□rt zu RESPONSE- der IEC-Parametertabelle *)
RD_BOOL_BY_SYM_1(
        IN              :=      ServiceStr,
        ERR             :=      RdBySymErr);
(* Variable nicht vorhanden *)
if RdBySymErr > 0 then
        ServiceErr      :=      true;
        ServiceMsg      :=      CONCAT('The Service ', ServiceName);
        ServiceMsg      :=      CONCAT(ServiceMsg, ' is not supplied by this Dataobject');
end_if;
(* Flag f□r diesen Service ist false *)
if RD_BOOL_BY_SYM_1.OUT = false and ServiceErr = false then
        ServiceErr      :=      true;
        ServiceMsg      :=      CONCAT('The Service ', ServiceName);
```

```
        ServiceMsg            :=    CONCAT(ServiceMsg, ' is not activated');
    end_if;

(*''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''*)

    (* Init *)
    Cnt                              :=    1;

    (* Ausf□hrung des Services GetDataValues *)

    if ServiceErr = false and EQ_STRING(ServiceName, 'GetDataValues') then
        (* Anzahl der hier m□glichen Typkonvertierungen *)
        ConvCnt                      :=    3;
        (* Schleifenbedingung vorbereiten *)
        RdBySymErr            :=    10;

        (* GetDataValues der Data Class, IEC 61850-7-2, S 50 *)
```

```
while Cnt <= ConvCnt and RdBySymErr > 0 do
    case Cnt of
    (* Versuch, den Wert als INT zu lesen *)
    1:    TmpServiceMsg    :=    INT_TO_STRING(
                                    RD_INT_BY_SYM(ParameterStr, RdBySymErr),
                                    '%d');
    (* Versuch, den Wert als REAL zu lesen *)
    2:    TmpServiceMsg    :=    REAL_TO_STRING(
                                    RD_REAL_BY_SYM(ParameterStr, RdBySymErr),
                                    '%f');
    (* Versuch, den Wert als STRING zu lesen *)
    3:    TmpServiceMsg    :=    RD_STRING_BY_SYM(ParameterStr, RdBySymErr);
    end_case;

    (* Bei fehlerfreier Wertermittlung Ergebnis sichern *)
    if RdBySymErr = 0 then
        ServiceMsg    :=    TmpServiceMsg;
```

```
                end_if;
                Cnt                         :=      Cnt + 1;
            end_while;

            (* Fehlermeldung bei nicht ermitteltem Datentyp *)
            if EQ_STRING(ServiceMsg, '') then
                ServiceMsg          :=      'Service could not determine datatyp';
            end_if;

        end_if;

(*''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''*)

        (* Ausf□hrung des Services SetDataValues *)

        if ServiceErr = false and EQ_STRING(ServiceName, 'SetDataValues') then
```

```
(* Parameterstring in Attribut und Wert auftrennen *)
ParamAttr                   :=      LEFT(ParameterStr,FIND(ParameterStr,',') - 1);
ParaValue                   :=      RIGHT(ParameterStr,
                                        len(ParameterStr) - FIND(ParameterStr,','));

(* Anzahl der hier m□glichen Typkonvertierungen *)
ConvCnt                             :=      3;
(* Schleifenbedingung vorbereiten *)
WrBySymErr                  := 10;

(* SetDataValues der Data Class, IEC 61850-7-2, S 52 *)
while Cnt <= ConvCnt and WrBySymErr > 0 do
        case Cnt of
        (* Versuch, den Wert als INT zu schreiben *)
        1:      WrBySymErr :=       WR_INT_BY_SYM(
                                        ParamAttr,
                                        STRING_TO_INT(ParaValue));
```

```
                    (* Versuch, den Wert als REAL zu schreiben *)
                    2:      WrBySymErr :=       WR_REAL_BY_SYM(
                                                    ParamAttr,
                                                    STRING_TO_REAL(ParaValue));
                    (* Versuch, den Wert als STRING zu schreiben *)
                    3:      WrBySymErr :=       WR_STRING_BY_SYM(
                                                    ParamAttr,
                                                    ParaValue);
                    end_case;
                    Cnt                 :=      Cnt + 1;
            end_while;

    end_if;

(*''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''*)

    (* Ausf□hrung des Services GetLDDirectory *)
```

```
if      ServiceErr = false and
        (
        EQ_STRING(ServiceName, 'GetDataDefinition') or
        EQ_STRING(ServiceName, 'GetDataDirectory') or
        EQ_STRING(ServiceName, 'GetLNDirectory') or
        EQ_STRING(ServiceName, 'GetLDDirectory') or
        EQ_STRING(ServiceName, 'GetServerDirectory')
        )
        then

        (*
        ParameterStr zu Variablenreferenz umbauen
        R□ckgabewert mit Variablenreferenz ermitteln
        Datei mit erfragten Infos ergibt sich aus ServiceName - 3 Zeichen
        *)
        ParameterStr            :=      CONCAT(ParameterStr, '.');
```

```
        TmpString           :=      DELETE(ServiceName, 3 , 1);
        ParameterStr     :=      CONCAT(ParameterStr, TmpString);
        (* R□ckgabewert mit Variablenreferenz ermitteln *)
        TmpString           :=      RD_STRING_BY_SYM(ParameterStr, RdBySymErr);

        (* Bei fehlgeschlagenem Lesen Fehlermeldung erzeugen *)
        if RdBySymErr = 0 then
                ServiceMsg      :=      TmpString;
        else
                ServiceErr      :=      true;
                ServiceMsg      :=      CONCAT('Execution of Service ', ServiceName);
                ServiceMsg      :=      CONCAT( ServiceMsg, ' failed');
        end_if;
    end_if;

(*''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''*)
```

```
        (* Rücksendung der Antwort vorbereiten *)


        (* Datenl□nge ermitteln *)
        TcpOutStruc.Len                 :=      len(ServiceMsg);
        (* Umkopieren der Antwort in einen Puffer f□r den Versand *)
        (* Keine Auswertung der Rueckgabewertes in BufCpy von STRING_TO_BUFFER*)
        BufCpy                                  :=      STRING_TO_BUFFER(
                                                        ServiceMsg,                     (* Kopierquelle *)
                                                        TcpOutStruc.SndChr[0],  (* Kopiersenke *)
                                                        TcpOutStruc.Len);               (* Datenlaenge*)


        (* Startberechtigung f□r Zeichenversand im n□chsten Zyklus *)
        StartDataSending                :=      true;
end_if;


(*
##########################################################################################
```

AUSGABE:R□cksendung der Antwort, erfolgt erstmal ohne Differenzierung des Inhalts

##

*)

```
IP_USEND_1(
REQ                                 :=      StartDataSending,
ID                                  :=      IpConId,
SD_1                        :=      TcpOutStruc);
IpSndDone                   :=      IP_USEND_1.DONE;
IpSndErr                    :=      IP_USEND_1.ERROR;
IpSndStat                   :=      IP_USEND_1.STATUS;
TcpOutStruc                 :=      IP_USEND_1.SD_1;

(* Flag bei Abschluss Zeichenversand r□cksetzen *)
if IpSndDone then
        StartDataSending            :=      false;
end_if;
```

```
(*
##########################################################################################
*)
(*@KEY@: END_WORKSHEET *)
END_FUNCTION_BLOCK
```

Quellcode des Datenmodells aus Abschnitt 4.5.5

```
(*@PROPERTIES_EX@
TYPE: DATA_TYPE
LOCALE: 0
*)
TYPE

(*
##########################################################################################
Allgemeine Datentypen
```

```
##########################################################################################
*)

    ARR_B_0_1460                :    ARRAY[0..1460] OF BYTE;
    ARR_STR_0_100               :    ARRAY[0..100] OF STRING;

    UDT_TcpIn                   :
    STRUCT
    Len                             :    int;              (* TCP-Telegrammlnge *)
    RsvChr                          :    ARR_B_0_1460;     (* empfangene Textmitteilung *)
    END_STRUCT;

    UDT_TcpOut                  :
    STRUCT
    Len                             :    int;              (* TCP-Telegrammlnge *)
    SndChr                          :    ARR_B_0_1460;     (* versendete Textmitteilung *)
    END_STRUCT;
```

```
        UDT_str_255                     :       String(255);        (* lange Objektreferenzen usw. *)
        UDT_str_1000                    :       String(1000);       (* Liste langer Objektreferenzen *)

(*
##########################################################################################
Datenmodell
##########################################################################################
*)

(* Common Data Attribute (CDA) *)

        UDT_AnalogueValue               :                           (* IEC 61850-7-3, S.16, Table 2 *)
        STRUCT
        i                               :       int;                (* Auswahl M *)
        f                               :       real;               (* Auswahl M, IEC: FLOAT32 *)
        END_STRUCT;
```

```
UDT_Vector_001          :                       (* IEC 61850-7-3, S.19, Table 10 *)
STRUCT
mag                     :   UDT_AnalogueValue;
ang                     :   UDT_AnalogueValue;
END_STRUCT;

(*''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''*)

(* Common Data Classes (CDC) *)

UDT_CMV_001             :                       (* IEC 61850-7-3, S.19, Table 10 *)
STRUCT
instCVAL                :   UDT_Vector_001;
GetDataValues           :   bool;               (* Serviceauswahl M *)
SetDataValues           :   bool;               (* Serviceauswahl M/O *)
GetDataDirectory        :   bool;               (* Serviceauswahl M/O *)
```

```
GetDataDefinition       :   bool;           (* Serviceauswahl M/O *)
DataDirectory           :   string;         (* GetDataDirectory, Reihenfolge beachten*)
DataDefinition          :   string;         (* GetDataDefinition, Reihenfolge beachten *)
END_STRUCT;

UDT_WYE_001             :                   (* IEC 61850-7-3, S.30, Table 25 *)
STRUCT
(* Daten *)
d                       :   UDT_str_255;    (* IEC: VISIBLESTRING255 *)
phsA                    :   UDT_CMV_001;
(* Metadaten *)
GetDataValues           :   bool;           (* Serviceauswahl M *)
SetDataValues           :   bool;           (* Serviceauswahl M/O *)
GetDataDirectory        :   bool;           (* Serviceauswahl M/O *)
GetDataDefinition       :   bool;           (* Serviceauswahl M/O *)
DataDirectory           :   string;         (* GetDataDirectory, Reihenfolge dirObjType beachten*)
DataDefinition          :   string;         (* GetDataDefinition, Reihenfolge dirObjRef beachten *)
```

```
        END_STRUCT;

(*''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''*)

(* Logical Nodes (LN) *)

        UDT_WGEN_002                :                                     (* IEC 61400-25-2, S.21, Table 10 *)
        STRUCT
        (* Daten *)
        W                               :       UDT_WYE_001;
        (* Metadaten *)
        GetLNDirectory              :       bool;                         (* Serviceauswahl M/O *)
        LNDirectory                 :       string;                       (* Auflistung der untergeordneten DA *)
        END_STRUCT;

(*''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''*)
```

```
(* Logical Devices (LD) *)

    UDT_LD_WEA08        :                               (* IEC 61850-7-2, S.34, Table 14 *)
    STRUCT
    (* Daten *)
    WGEN2                   :   UDT_WGEN_002;
    (* Metadaten *)
    GetLDDirectory      :   bool;                   (* Serviceauswahl M/O *)
    LDDirectory         :   string;                 (* Auflistung der untergeordneten LN *)
    END_STRUCT;

(*''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''*)

(* Physical Devices (PHD) *)

    UDT_Server_001      :                               (* Server Class IEC 61850-7-2, S.24 *)
    STRUCT
```

```
    (* Daten *)
    WEA08                              :      UDT_LD_WEA08;
    (* Metadaten *)
    GetServerDirectory        :      bool;                (* Serviceauswahl M/O *)
    ServerDirectory               :      string;               (* Auflistung der untergeordneten LD *)
    END_STRUCT;

(*''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''*)

END_TYPE
```

Erklärung zur selbständigen Anfertigung

Ich erkläre, dass ich die vorliegende Arbeit selbständig und nur unter Verwendung der angegebenen Literatur und Hilfsmittel angefertigt habe.

Bad Pyrmont, 12.03.2009